تم تنزيل الكتب والبرامج و الفيديو من مكتبة فرحات

مكتبة فرحات الهندسية



www.farahat-library.com/blog

لتحميل أفضل افلام الكارتون الهادفة

مكتبة فرحات المرئية



www.farahat-library.com/cartoon

لتحميل البرامج المفيدة و المنتجه

مكتبة فرحات للبرمجيات



www.farahat-library.com/software

لبيع وشراء المنتجات الغير متوفرة في الاسواق

مول فرحات



www.farahat-library.com/mall

المملكـــة العربيـــة الســـعودية

المؤسسة العامة للتعليم الفني والتدريب المهني

الإدارة العامة لتصميم وتطوير المناهج

البرمجيات

برمجة ٢

١٤٢ حاب

مقدمه

الحمد للّه وحده، والصلاة والسلام على من لا نبي بعده، محمد وعلى آله وصحبه، وبعد:

تسعى المؤسسة العامة للتعليم الفني والتدريب المهني لتأهيل الكوادر الوطنية المدربة القادرة على شغل الوظائف التقنية والفنية والمهنية المتوفرة في سوق العمل، ويأتي هذا الاهتمام نتيجة للتوجهات السديدة من لدن قادة هذا الوطن التي تصب في مجملها نحو إيجاد وطن متكامل يعتمد ذاتياً على موارده وعلى قوة شبابه المسلح بالعلم والإيمان من أجل الاستمرار قدماً في دفع عجلة التقدم التنموي؛ لتصل بعون اللّه تعالى لمصاف الدول المتقدمة صناعياً.

وقد خطت الإدارة العامة لتصميم وتطوير المناهج خطوة إيجابية تتفق مع التجارب الدولية المتقدمة في بناء البرامج التدريبية، وفق أساليب علمية حديثة تحاكي متطلبات سوق العمل بكافة تخصصاته لتلبي متطلباته ، وقد تمثلت هذه الخطوة في مشروع إعداد المعايير المهنية الوطنية الذي يمثل الركيزة الأساسية في بناء البرامج التدريبية، إذ تعتمد المعايير في بنائها على تشكيل لجان تخصصية تمثل سوق العمل والمؤسسة العامة للتعليم الفني والتدريب المهني بحيث تتوافق الرؤية العلمية مع الواقع العملي الذي تفرضه متطلبات سوق العمل، لتخرج هذه اللجان في النهاية بنظرة متكاملة لبرنامج تدريبي أكثر التصاقاً بسوق العمل، وأكثر واقعية في تحقيق متطلباته الأساسية.

وتتناول هذه الحقيبة التدريبية " برمجة ٢ " لمتدربي قسم" البرمجيات " للكليات التقنية موضوعات حيوية تتناول كيفية اكتساب المهارات اللازمة لهذا التخصص.

والإدارة العامة لتصميم وتطوير المناهج وهي تضع بين يديك هذه الحقيبة التدريبية تأمل من اللّه عز وجل أن تسهم بشكل مباشر في تأصيل المهارات الضرورية اللازمة، بأسلوب مبسط يخلو من التعقيد، وبالاستعانة بالتطبيقات والأشكال التي تدعم عملية اكتساب هذه المهارات.

واللّه نسأل أن يوفق القائمين على إعدادها والمستفيدين منها لما يحبه ويرضاه؛ إنه سميع مجيب الدعاء.

الإدارة العامة لتصميم وتطوير المناهج



المملكـة العربيــة الســعودية
المؤسسة العامة للتعليم الفني والتدريب المهني
الإدارة العامة لتصميم وتطوير المناهج

برمجة ٢

المصفوفـــات

المصفوفـــات

١

الجـــــدارة:

معرفة كيفية استخدام المصفوفات لحل بعض المشاكل البرمجية.

الأهـــــداف:

عندما تكمل هذه الوحدة تكون قادراً على:

١ - معرفة الغاية من استخدام المصفوفات.
٢ - تعريف المصفوفات وحجز المواقع لها.
٣ - إعطاء المصفوفات القيم الابتدائية عند التعريف.
٤ - الوصول لموقع معين داخل المصفوفة لتعديل محتوياته.
٥ - ترتيب عناصر المصفوفات.
٦ - معرفة طرق البحث عن عنصر معين داخل المصفوفات.
٧ - التعامل مع المصفوفات ذات البعدين.
٨ - كتابة البرامج التي تستخدم المصفوفات لحل المشاكل البرمجية.

مستوى الأداء المطلوب:

أن يصل المتدرب إلى إتقان هذه الجدارة بنسبة ١٠٠٪.

الوقت المتوقع للتدريب: ١٠ ساعات.

الوسائل المساعـدة:

- قلم.
- دفتر.
- جهاز حاسب آلي.

متطلبات الجدارة:

اجتياز جميع الحقائب السابقة.

مقدمـة :

في هذه الوحدة سوف يتم التطرق للمصفوفات ذات البعد الواحد والمصفوفات ذات البعدين، حيث سنقوم بشرح كيفية تعريف المصفوفات وحجز المواقع لها مع توضيح كيفية إعطاء القيم الابتدائية للمصفوفات عند تعريفها. كما وسنقوم بشرح عمليات ترتيب عناصر المصفوفات والبحث عن عنصر معين أو عدة عناصر في المصفوفات. وفي نهاية هذا الفصل هنالك عدد من التمارين المتعلقة بالمصفوفات.

تعريف المصفوفات وحجز المواقع لها :

المصفوفات هي عبارة عن مواقع يتم تخزين البيانات فيها لمدة مؤقتة (طيلة فترة تنفيذ البرنامج فقط)، وعند تعريف المصفوفة وإنشاءها يتم حجز عدد محدد من المواقع المتجاورة في الذاكرة لتخزين البيانات فيها، حيث يتم الوصول للبيانات المخزنة في هذه المواقع عن طريق اسم المصفوفة ورقم الموقع (Index). والغاية من استخدام المصفوفات هي تخزين عدد غير محدد من القيم تحت اسم واحد فقط (اسم المصفوفة) دون الحاجة إلى تخزين كل قيمة في متغير (Variable) منفصل.

لاستخدام المصفوفات في البرنامج لابد من تعريفها وحجز المواقع لها. حيث يتم ذلك كما يلي:

```
1. int array1[];
2. array1[] = new int[9];
```

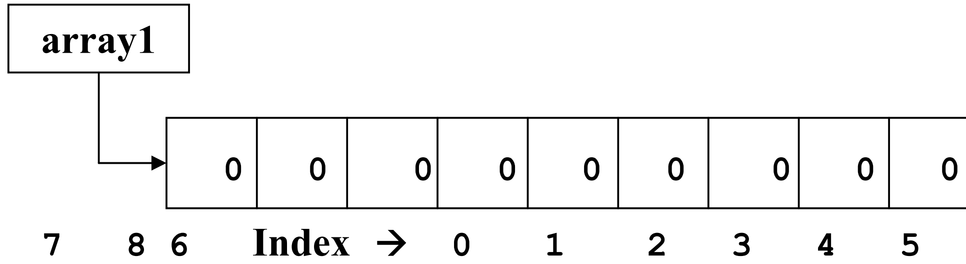
في السطر رقم (١) تم تعريف المصفوفة array1 من نوع int ، أي أننا نستطيع تخزين أعداد من نوع int في هذه المصفوفة. بينما في السطر رقم (٢) تم حجز ٩ مواقع لهذه المصفوفة (من الموقع رقم صفر إلى الموقع رقم ٨) لنستطيع تخزين ٩ أعداد صحيحة على الأكثر في هذه المصفوفة. كما ويمكن دمج الجملتين السابقتين بجملة واحدة لتصبح كما يلي:

```
int array1[] = new int[9];
```

ويمكن كتابة الجملة السابقة بالشكل التالي:

```
int[] array1 = new int[9];
```

يتم حجز المواقع للمصفوفة array1 كما في الشكل (١-١):



شكل (١-١)

في لغة جافا، رقم موقع العنصر في المصفوفة يكتب بين أقواس مربعة بعد اسم المصفوفة (مثال: array1[k]، حيث أن k يمثل رقم الموقع في المصفوفة، وفي مثالنا السابق هو عدد صحيح محصور بين الصفر والثمانية). وبشكل عام، عند حجز n من المواقع للمصفوفة فإن أرقام هذه المواقع تكون من صفر ولغاية n-1.

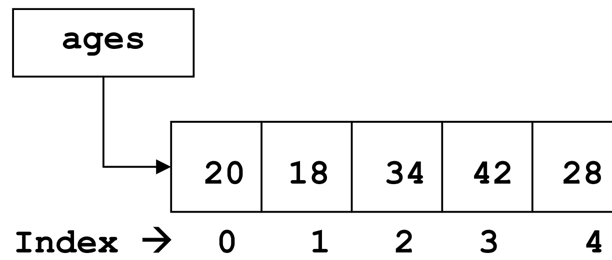
ونستطيع بشكل اختياري أن نحدد للمصفوفة قيماً ابتدائية يتم تحديدها عند تعريف المصفوفة، وإذا لم نحدد للمصفوفة قيماً ابتدائية فإنه يتم تخزين القيمة التلقائية (Default Value) لنوع المصفوفة وذلك عند حجز المواقع لها. والقيم التلقائية للأنواع هي كما يلي:

int, byte, short, long	→	0
double, float	→	0.0
char	→	\u0000 فراغ
String	→	null
Boolean	→	false

ويمكن تحديد القيم الابتدائية للمصفوفة بالطريقة التالية:

```
int ages[]= {20, 18, 34, 42, 28};
```

من خلال هذه الجملة قمنا بتعريف مصفوفة اسمها ages، وخزّنا فيها قيماً ابتدائية، حيث سيتم حجز مواقع على عدد هذه القيم الابتدائية. والشكل (١-٢) يوضح عملية التخزين.



شكل (١-٢)

وللوصول للرقم 42 في المصفوفة ages يجب استخدام الشكل التالي: ages[3]، حيث نستطيع طباعة الرقم 42 عن طريق الجملة التالية:

```
System.out.println(ages[3]);
```

ولتعديل القيمة المخزّنة في الموقع رقم 1 لتصبح 53 عوضاً عن 18، يجب تنفيذ الجملة التالية:

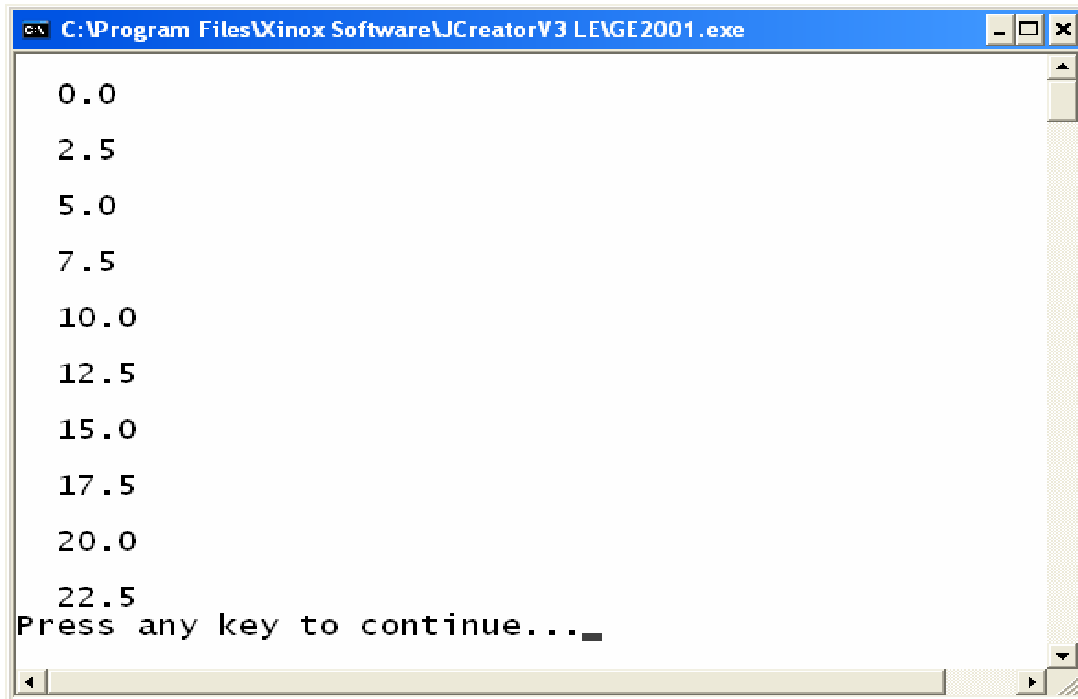
```
ages[1]=53;
```

مثال: ١-١:

```
// array1.java
public class array1{
 public static void main(String args[]){
  double a[]=new double[10];
  for(int i=0; i<10; i++){
   a[i]=i*2.5;
   System.out.println(a[i]);
  } // end for
 } // end main
} // end class array1
```

شرح المثال:

في السطر رقم (٣) قمنا بتعريف مصفوفة اسمها a من نوع double وحجزنا لها ١٠ مواقع. وفي السطر رقم (٥) تم تخزين ناتج العملية التالية في مواقع المصفوفة i*2.5 حيث تتغير قيمة i من صفر ولغاية تسعة لتحدد رقم الموقع المراد تخزين ناتج العملية فيه ولتؤثر على ناتج العملية. وفي السطر رقم (٦) قمنا بطباعة محتويات المصفوفة a. والشكل (١-٣) يبين نتائج البرنامج السابق:



شكل (١-٣)

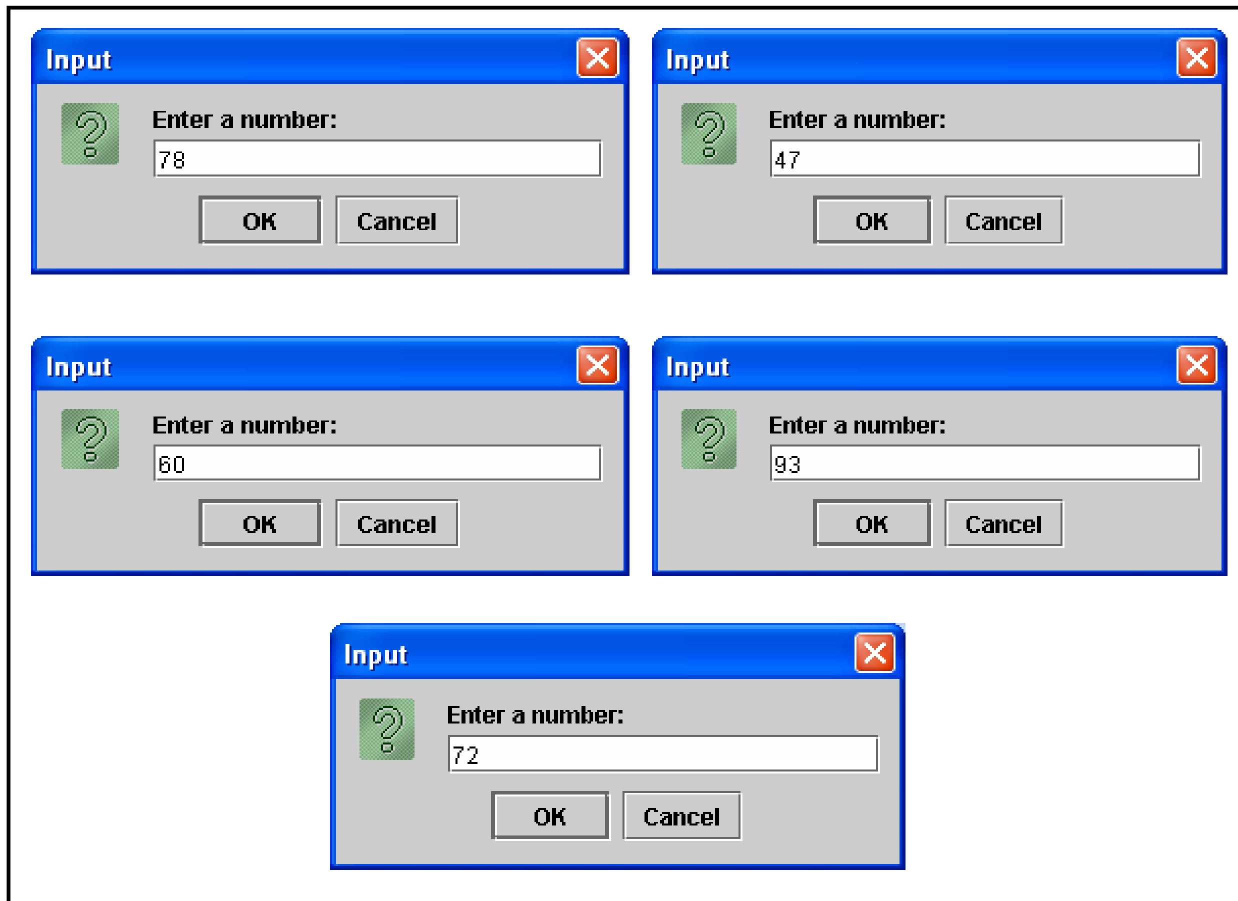
مثال: ١-٢:

```
// array2.java
import javax.swing.*;
 public class array2{
  public static void main(String args[]){
  int b[]=new int[5];
  String s;
  for(int i=0; i<5; i++){
   s=JOptionPane.showInputDialog("Enter a number:");
   b[i]=Integer.parseInt(s);
  } // end for
  for(int i=0; i<5; i++)
```

```
11.    if(b[i]>=60)
12.      System.out.println(b[i]);
13.   } // end main
14.  } // end class array2
```

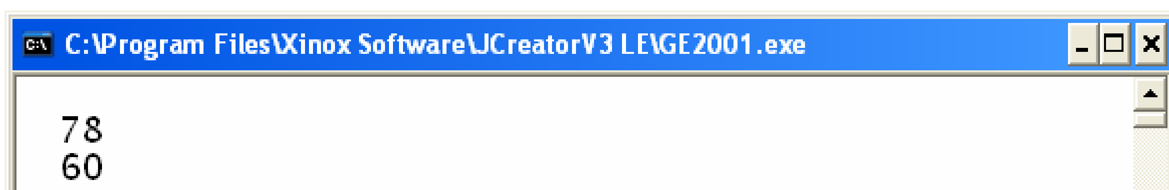
شرح المثال:

في السطر رقم (٤) قمنا بتعريف مصفوفة اسمها b وتم حجز ٥ مواقع لهذه المصفوفة. في الأسطر (٦-٩) يتم إدخال قيم ليتم تخزينها في المصفوفة b. وفي الأسطر (١٠-١٢) تتم عملية طباعة محتويات المصفوفة b. حيث يقوم هذه البرنامج بطلب المستخدم إدخال ٥ درجات لتخزينها في المصفوفة b، بعد ذلك يقوم البرنامج بطباعة الدرجات التي تزيد عن أو تساوي ٦٠. والشكل (١-٤) تبين عمليات إدخال الدرجات:



شكل (١-٤)

بينما يبين الشكل (١-٥) نتائج البرنامج السابق:



شكل (١-٥)

ملاحظـات مهمـة:

١ - يجب أن يكون رقم الموقع (Index) عند التعامل مع المصفوفة عدداً صحيحاً موجباً.

٢ - يجب أن لا نتجاوز عدد المواقع المحجوزة للمصفوفة عند استخدامها.

٣ - إذا لم نحدد قيماً ابتدائية للمصفوفة فيجب أن نستخدم الكلمة المحجوزة (new) لحجز مواقع للمصفوفة كما ذكر سابقاً.

٤ - إذا لم تعطى المصفوفة قيماً ابتدائية عند تعريفها فإنها تأخذ القيم التلقائية (Default Value) كقيم ابتدائية وذلك حسب النوع (Type) الذي حدد للمصفوفة. كما ذكر سابقاً.

٥ - نستطيع معرفة عدد المواقع المحجوزة للمصفوفة من خلال كتابة اسم المصفوفة ثم نقطة ثم length. (مثال: array1.length ، من خلال هذه الجملة نستطيع معرفة عدد المواقع المحجوزة للمصفوفة array1).

٦ - نستطيع استخدام أحد الأشكال التالية لتعريف وحجز مواقع المصفوفة:

```
1. int array[]=new int[5];
```

```
2. int array[];
  array=new int[5];
```

```
3. int [] array = new int[5];
```

```
4. int [] array;
   array=new int[5];
```

٧ - نستطيع استخدام أحد الأشكال التالية لتخزين القيم الابتدائية في المصفوفة:

```
1. int array[]={5, 3, 8, 9, 2};
```

```
2. int array[]=new int[] {5, 3, 8, 9, 2};
```

أمثلة على استخدام المصفوفات:

مثال: ١-٣:

```
//array3.java
import javax.swing.*;
class array3{
  public static void main(String args[]){
   String s, output, title, str1, str2;
   str1="Enter how many number you want to store in the array:";
   str2="Enter number # ";
   int n, odd=0;
   s=JOptionPane.showInputDialog(str1);
   n=Integer.parseInt(s);
   int [] arr=new int[n];
   output= " ";
   for(int i=0; i<arr.length; i++){
    s=JOptionPane.showInputDialog(str2+(i+1)+":");
    arr[i]=Integer.parseInt(s);
    output+=arr[i]+"\n ";
   }  //end for
   for(int i=0; i<arr.length; i++)
   if(arr[i]%2==1) odd++; // end for
   title="The results of the example (1-3)";
   output+="\nThere are "+odd+" odd numbers in the array";
   JOptionPane.showMessageDialog(null, output, title,
          JOptionPane.INFORMATION_MESSAGE);
   System.exit(0);
  }       //end main
} //end class array3
```

شرح المثال:

في الأسطر (٨-٩) يطلب البرنامج إدخال رقم، حيث يحوّل هذا العدد إلى رقم صحيح ويخزّن في المتغير n. في السطر رقم (١٠) يتم تعريف المصفوفة arr وحجز عدد n من المواقع لهذه المصفوفة (أي يتم تحديد عدد العناصر المحجوزة للمصفوفة عن طريق المستخدم للبرنامج). في الأسطر (١٢-١٦) يتم إدخال n من الأرقام وتخزّن في المصفوفة arr (لاحظ الدوران يبدأ من صفر ولغاية أقل من arr.length). في السطر رقم (١٨) يتم فحص الأرقام المخزنة في المصفوفة arr، حيث إذا كان الرقم فرديا يضاف واحد للمتغير odd (متغير يخزّن فيه عدد الأرقام الفردية) . وفي السطر رقم (٢١) يتم طباعة نتائج البرنامج. الشكل (١-٦) يبين عمليات الإدخال في البرنامج:

Input
Enter how many number you want to store in the array:
4
OK Cancel

Input
Enter number # 1:
34
OK Cancel

Input
Enter number # 2:
19
OK Cancel

Input
Enter number # 3:
21
OK Cancel

Input
Enter number # 4:
13
OK Cancel

شكل (١-٦)

بينما يعرض الشكل (١-٧) مخرجات البرنامج السابق:

The results of the example (1-3)
34
19
21
13
There are 3 odd numbers in the array

شكل (١-٧)

مثال: ١-٤:

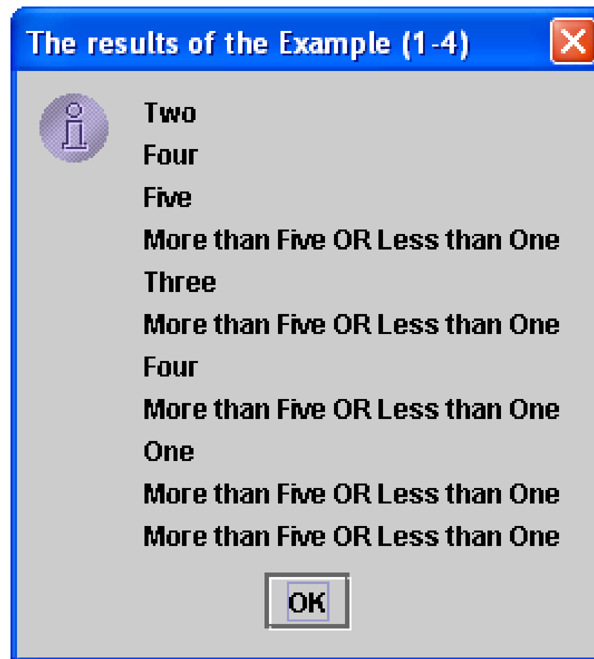
```
// array4.java

import javax.swing.*;
class array4{
 public static void main(String args[]){
  int a[]={2, 4, 5, -5, 3, 10, 4, 11, 1, 7, -2};
  String title="The results of the Example (1-4)";
  String results="";

  for(int i=0; i<a.length; i++)
  {
   switch(a[i]){
   case 1:  results=results+"One\n"; break;
   case 2:  results=results+"Two\n"; break;
   case 3:  results=results+"Three\n"; break;
   case 4:  results=results+"Four\n"; break;
   case 5:  results=results+"Five\n"; break;
   default: results=results+"More than Five OR Less than One\n";
   }
  }
  JOptionPane.showMessageDialog(null, results, title,
      JOptionPane.INFORMATION_MESSAGE);
  System.exit(0);
 }
}
```

شرح المثال:

في السطر رقم (٤) تم تعريف المصفوفة a وإعطائها قيماً ابتدائية. في الأسطر (٧-١٧) دوران للتعامل مع جميع عناصر المصفوفة من خلال جملة switch ، حيث سيتم طباعة الرقم بالأحرف إذا كان هذا الرقم بين واحد وخمسة وتطبع جملة "More than Five OR Less than One" إذا كان الرقم غير ذلك. والشكل (١-٨) يبين مخرجات البرنامج السابق:



شكل (١-٨)

مثال: ١-٥:

```
// array5.java

import javax.swing.*;
class array5{
 public static void main(String args[]){
  double marks[] = new double[6];
  String names[] = new String[6];
  String s;
  String t1 = "Enter the Student's name:";
  String t2 = "Enter his mark:";
```

```
9.      for(int i=0; i<6; i++){
10.      s=JOptionPane.showInputDialog(t1);
11.      names[i]=s;
12.      s=JOptionPane.showInputDialog(t2);
13.      marks[i]=Double.parseDouble(s);
14.     }
15.     String title = "The passed students";
16.   String results="The following students are passed the exam:\n";
17.     for(int i=0; i<6; i++){
18.      if(marks[i]>=60)
19.        results=results+names[i]+"\n";
20.     }
21.   JOptionPane.showMessageDialog(null, results,  title,
          JOptionPane.INFORMATION_MESSAGE);
22.     System.exit(0);
23.    }
24.   }
```

شرح المثال:

في السطر رقم (٤) تم تعريف مصفوفة اسمها marks من نوع double وحجز ٦ مواقع لها، حيث ستخزن الدرجات في هذه المصفوفة. بينما في السطر رقم (٥) تم تعريف مصفوفة اسمها names من نوع String وحجز ٦ مواقع لها حيث ستستخدم هذه المصفوفة لتخزين أسماء الطلاب. في الأسطر (٩-١٤) استخدم الدوران لإدخال أسماء ودرجات الطلاب الستة وتخزينها في المصفوفات الخاصة بها. في الأسطر (١٧-٢٠) ومن خلال جملة الدوران يتم إضافة أسماء الطلاب الذين تزيد درجاتهم عن أو تساوي ٦٠ إلى المخرجات. وفي السطر (٢١) يتم طباعة المخرجات والتي تحتوي على أسماء الطلاب الذين تزيد درجاتهم عن أو تساوي ٦٠. والشكل (١-٩) يبين عمليات الإدخال في البرنامج:

Input
Enter the Student's name:
Ahmad
OK Cancel

Input
Enter his mark:
72
OK Cancel

Input
Enter the Student's name:
Saoud
OK Cancel

Input
Enter his mark:
47
OK Cancel

Input
Enter the Student's name:
Khaled
OK Cancel

Input
Enter his mark:
57
OK Cancel

Input
Enter the Student's name:
Saleh
OK Cancel

Input
Enter his mark:
45
OK Cancel

شكل (١-٩)

Input
Enter the Student's name:
Ali
OK Cancel

Input
Enter his mark:
53
OK Cancel

Input
Enter the Student's name:
Mohammed
OK Cancel

Input
Enter his mark:
92
OK Cancel

تكملة الشكل (١-٩)

والشكل (١-١٠) يبين مخرجات البرنامج السابق:

The passed students
The following students are passed the exam:
Ahmad
Mohammed
OK

شكل (١-١٠)

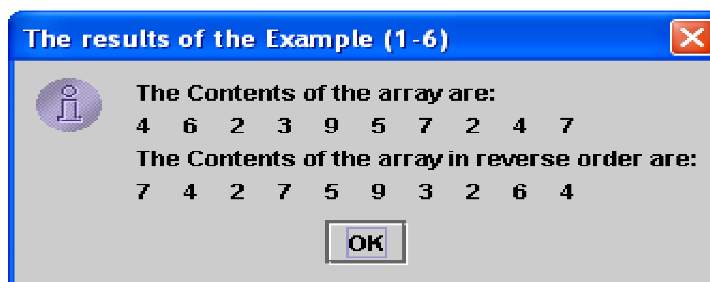
مثال: ١-٦:

```
// array6.java

1.     import javax.swing.*;
2.     class array6{
3.      public static void main(String args[]){
4.       int num[] = new int[]{4, 6, 2, 3, 9, 5, 7, 2, 4, 7};
5.       String output="The Contents of the array are:\n";
6.       String title="The results of the Example (1-6)";
7.       for(int i=0; i<=num.length-1; i++)
8.        output+=num[i]+"     ";
9.       output+="\nThe Contents of the array in reverse order are:\n";
10.      for(int i=num.length-1; i>=0; i--)
11.       output+=num[i]+"     ";
12.    JOptionPane.showMessageDialog(null, output,  title,
          JOptionPane.INFORMATION_MESSAGE);
13.      System.exit(0);
14.     }
15.    }
```

شرح المثال:

في الأسطر (١٠-١١) تم إحداث دوران عكسي وذلك لإضافة عناصر المصفوفة إلى المخرجات وبشكل عكسي. حيث يقوم هذا البرنامج بطباعة محتويات المصفوفة num بشكل عادي وبشكل عكسي (بدءاً بالموقع الأخير في المصفوفة num.length-1 وانتهاءً بالموقع رقم صفر). والشكل (١-١١) يبين مخرجات البرنامج السابق:



شكل (١-١١)

ترتيب عناصر المصفوفة (Sorting):

في كثير من التطبيقات والبرامج قد نحتاج إلى ترتيب محتويات المصفوفات. وترتيب المصفوفات أما أن يكون ترتيباً تصاعدياً من الأصغر إلى الأكبر أو يكون تنازلياً من الأكبر إلى الأصغر. وهنالك عدد من خوارزميات الترتيب المستخدمة في ترتيب العناصر وسوف نستخدم الترتيب الفقاعي (Bubble Sort).

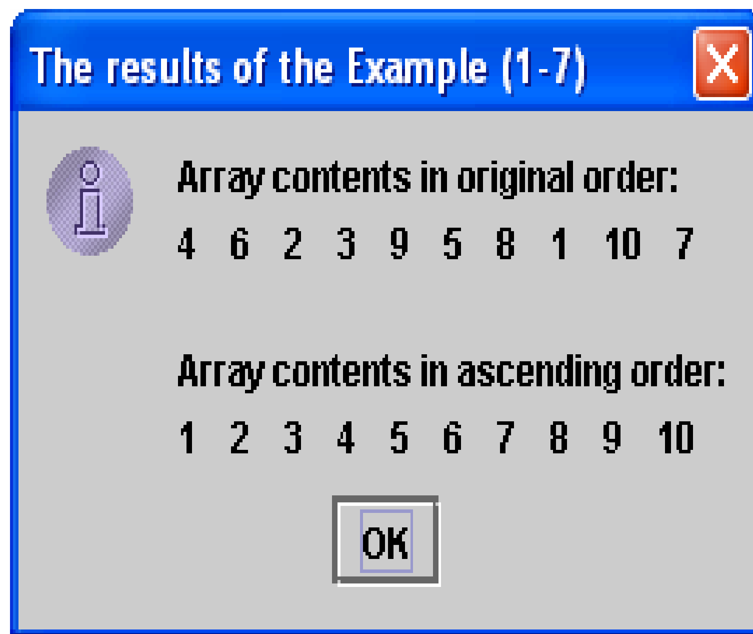
مثال ١-٧:

```
// array7.java

import javax.swing.*;
class array7{
 public static void main(String args[]){
  int num[] = new int[] {4, 6, 2, 3, 9, 5, 8, 1, 10, 7};
  int temp;
  String title="The results of the Example (1-7)";
  String output="Array contents in original order:\n";
  for(int i=0; i<num.length; i++)
   output+=num[i]+"    ";
  for(int i=1; i<num.length; i++)
   for(int j=0; j<num.length-1; j++)
    if(num[j]>num[j+1]){
     temp=num[j];
     num[j]=num[j+1];
     num[j+1]=temp;
    }
  output+="\n\nArray contents in ascending order:\n";
  for(int i=0; i<num.length; i++)
   output+=num[i]+"    ";
 JOptionPane.showMessageDialog(null, output, title,
        JOptionPane.INFORMATION_MESSAGE);
  System.exit(0);
 }
}
```

شرح المثال:

في الأسطر (١٠-١٦) هنالك عمليتا دوران متداخلتان يتم من خلالهما ترتيب عناصر المصفوفة num، حيث تم فحص شرط الترتيب في السطر رقم (١٢) وإذا تحقق هذا الشرط يتم تبديل عنصرين من عناصر المصفوفة بحيث يأخذ كل واحد من العنصرين مكان الآخر داخل المصفوفة. ومخرجات هذا البرنامج هي طباعة عناصر المصفوفة قبل الترتيب (من خلال الأسطر ٨-٩) وبعد الترتيب (من خلال الأسطر ١٨-١٩). هذا المثال يبين عملية الترتيب التصاعدي (Ascending). والشكل (١-١٢) يبين مخرجات البرنامج السابق:



شكل (١-١٢)

مثال ١-٨:

```
// array8.java

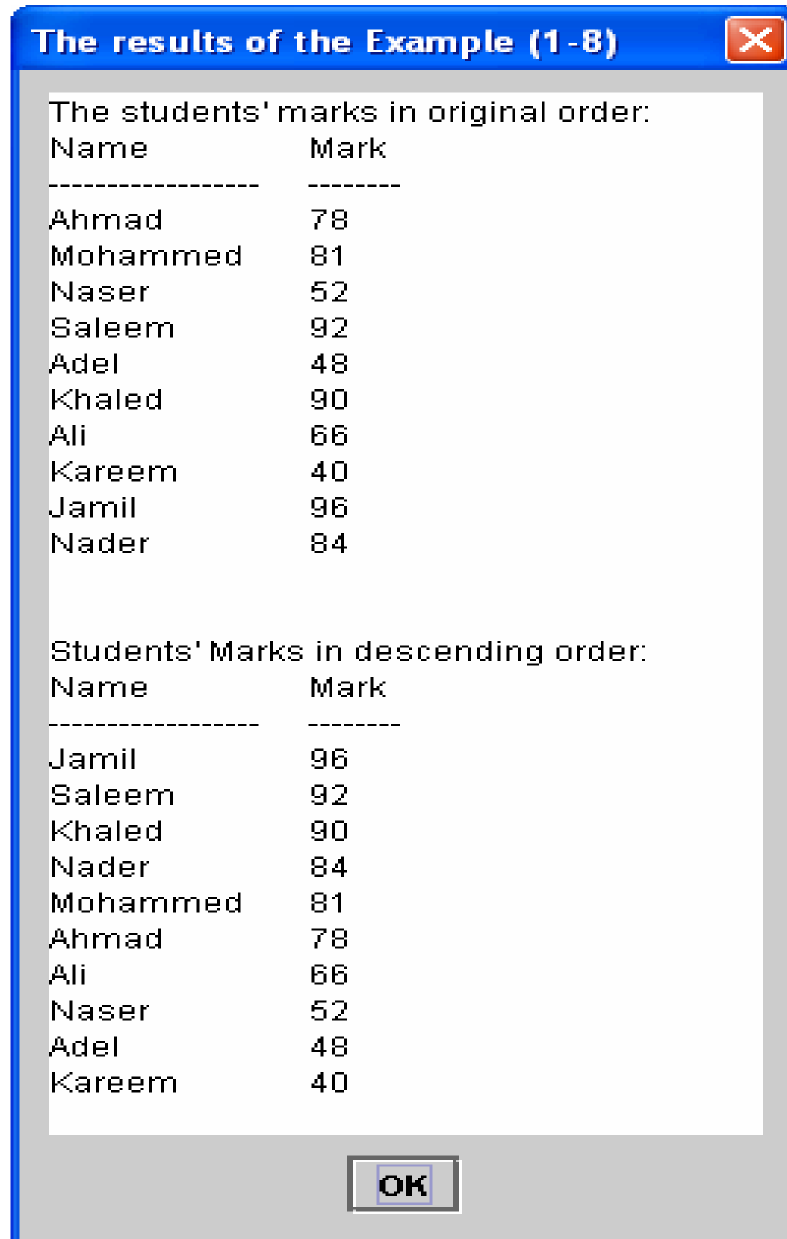
import javax.swing.*;
class array8{
 public static void main(String args[]){
  JTextArea outArea= new JTextArea();
    int mark[] = new int[] {78, 81, 52, 92, 48, 90, 66, 40, 96,84};
```

```
6.      String name[]= {"Ahmad", "Mohammed", "Naser", "Saleem",
                        "Adel", "Khaled", "Ali", "Kareem", "Jamil",
                        "Nader"};
7.      int temp_mark;
8.      String temp_name;
9.      String title="The results of the Example (1-8)";
10.     String output="The students' marks in original order:\n";
11.     output+="Name\tMark\n------------------\t--------\n";
12.     for(int i=0; i<mark.length; i++)
13.     output+=name[i]+"\t"+mark[i]+"\n";
14.     for(int i=1; i<mark.length; i++)
15.      for(int j=0; j<mark.length-1; j++)
16.       if(mark[j]<mark[j+1]){
17.         temp_mark=mark[j];
18.         mark[j]=mark[j+1];
19.         mark[j+1]=temp_mark;
20.         temp_name=name[j];
21.         name[j]=name[j+1];
22.         name[j+1]=temp_name;
23.       }
24.     output+="\n\nStudents' Marks in descending order:\n";
25.     output+="Name\tMark\n------------------\t--------\n";
26.     for(int i=0; i<mark.length; i++)
27.      output+=name[i]+"\t"+mark[i]+"\n";
28.     outArea.setText(output);
29.     JOptionPane.showMessageDialog(null, outArea, title,
            JOptionPane.PLAIN_MESSAGE);
30.     System.exit(0);
31.   }
32.  }
```

شرح المثال:

هذا المثال يبين عملية الترتيب التنازلي (Descending). في الأسطر (١٤-٢٣) تتم عملية ترتيب درجات الطلاب ترتيباً تنازلياً، حيث يتم من خلال الأسطر (١٧-١٩) تبديل الدرجات ليتم ترتيبها، وفي الأسطر

(٢٠-٢٢) يتم تبديل الأسماء لتبقى مرتبطة مع الدرجات الخاصة بها. والشكل (١-١٣) يبين مخرجات البرنامج السابق:



شكل (١-١٣)

البحث في المصفوفات (Searching):

عادةً يقوم المبرمج بالتعامل مع مصفوفات كبيرة الحجم وبالتالي لتحديد أي عنصر معين موجود في مصفوفة لابد من استخدام طرق البحث. من خلال هذا الدرس سوف نتعلّم طريقتين من طرق البحث وهما: البحث الخطي (linear Search) والبحث الثنائي (Binary Search).

يستخدم البحث الخطي للبحث عن عنصر معين داخل المصفوفة غير المرتبة أو المصفوفة المرتبة، وفي هذه الطريقة يتم مقارنة جميع محتويات المصفوفة مع القيمة المراد البحث عنها وبشكل متسلسل من بداية المصفوفة إلى نهايتها. بينما يستخدم البحث الثنائي للبحث عن عنصر معين داخل المصفوفة المرتبة فقط، وفي هذه الطريقة يتم تقسيم المصفوفة إلى نصفين وعن طريق المقارنة يتم تحديد إلى أي نصف ينتمي العنصر المراد البحث عنه وهكذا حتى يتم العثور على العنصر داخل المصفوفة إذا كان موجوداً. ويعتبر البحث الثنائي في المصفوفات المرتبة أسرع وأكفأ من البحث الخطي والأمثلة التالية توضح طرق البحث هذه.

مثال ١-٩:

```
// array9.java

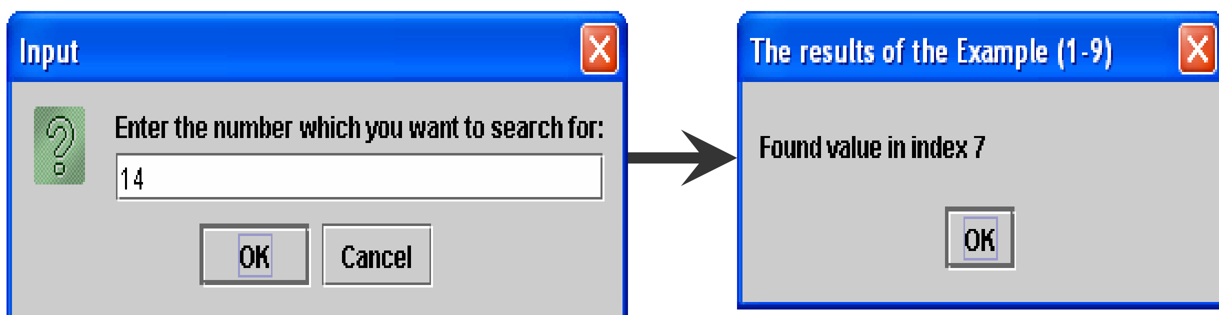
import javax.swing.*;
class array9{
 public static void main(String args[]){
  int n[] = new int[10];
  int num, k=-1;
  String title="The results of the Example (1-9)";
  String s, output="";

  for(int i=0; i<n.length; i++)
   n[i]=i*2;
s=JOptionPane.showInputDialog("Enter the number which you
                                          want to search for:");
  num=Integer.parseInt(s);
  for(int i=1; i<n.length; i++)
   if(n[i]==num){
    k=i;
```

```
15.        break;
16.       }
17.      if(k!=-1)
18.       output+="Found value in index "+k;
19.      else
20.       output+="Value not found";
21.      JOptionPane.showMessageDialog(null, output, title,
                                                  JOptionPane.PLAIN_MESSAGE);
22.      System.exit(0);
23.     }
24.   }
```

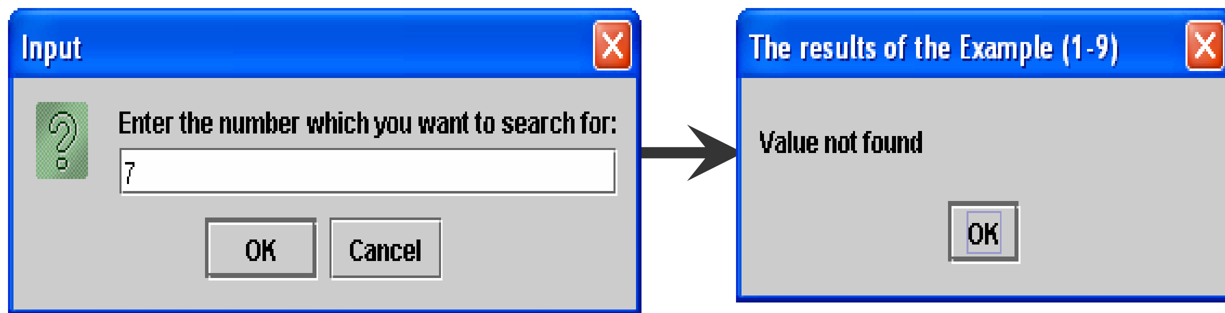
شرح المثال:

في هذا المثال تم استخدام طريقة البحث الخطي للبحث داخل مصفوفة مرتبة عن رقم معين يتم إدخاله عن طريق لوحة المفاتيح. حيث تقوم الأسطر (١٢-١٦) بمقارنة الرقم المراد البحث عنه والمخزن في المتغير num مع جميع محتويات المصفوفة، وفي حالة تم العثور على هذا الرقم في المصفوفة يتم تخزين موقعه في المتغير k، وفي حالة عدم العثور على الرقم في المصفوفة يبقى محتوى المتغير k كما هو -١. ومن خلال الأسطر (١٧-٢٠) يتم طباعة رسالة بعدم وجود الرقم المراد البحث عنه في المصفوفة إذا كانت قيمة k لم تتغير وبقيت -١، بينما إذا تغيرت قيمة k فهذا يعني بأن الرقم المراد البحث عنه موجود داخل المصفوفة وفي الموقع k وسوف يتم طباعة رسالة بذلك. والشكل (١-١٤) يبين تنفيذ البرنامج السابق في حالة العثور على الرقم ١٤ في المصفوفة:



شكل (١ -١٤)

بينما يبين الشكل (١-١٥) تنفيذ البرنامج في حالة عدم العثور على الرقم ٧ في المصفوفة:



شكل (١ -١٥)

مثال ١-١٠:

```
// array10.java

import javax.swing.*;
class array10{
 public static void main(String args[]){
  int id[] = new int[] {2, 10, 1, 7, 4, 6, 3, 8, 5, 9};
    String  name[]= {"Ahmad",  "Mohammed",  "Naser",  "Saleem",
                     "Adel", "Khaled", "Ali", "Kareem", "Jamil",
                     "Nader"};

  String s, stdName, title="The results of the Example (1-10)";
  String output="The student's name is:--> ";
  int no, index=-1;
    s=JOptionPane.showInputDialog("Enter the student's ID to
                                   display his name:");
  no=Integer.parseInt(s);

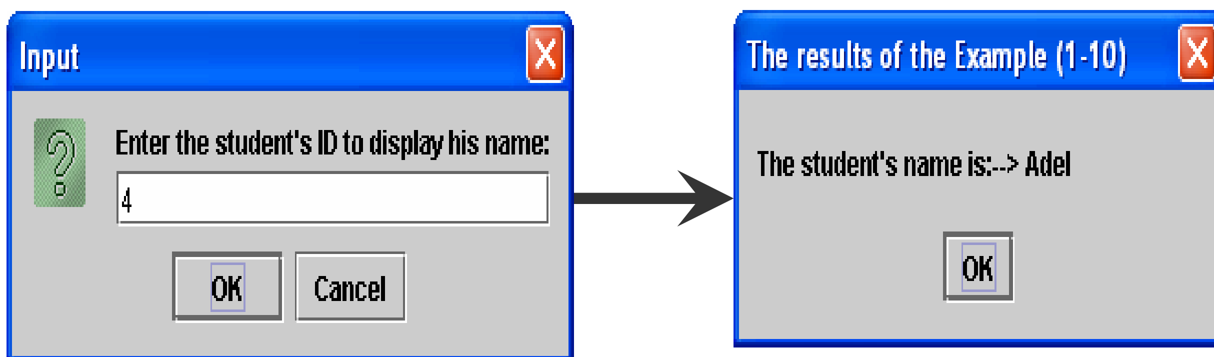
  for(int i=0; i<id.length; i++)
   if(id[i]==no){
    index=i;
    break;
   }
  if(index!=-1)
   output+=name[index];
```

```
18.     else
19.      output="There is no student with this ID !!!";
20.       JOptionPane.showMessageDialog(null, output, title,
                                       JOptionPane.PLAIN_MESSAGE);
21.     System.exit(0);
22.    }
23.   }
```

شرح المثال:

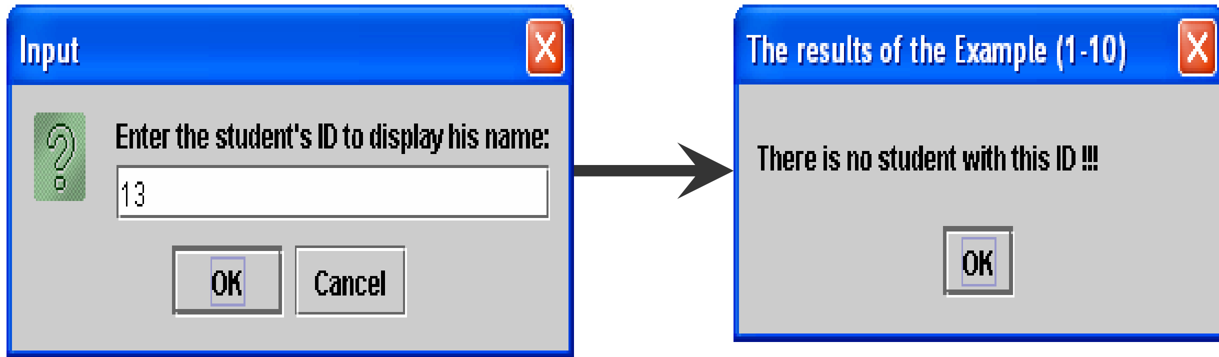
في هذا المثال تم استخدام طريقة البحث الخطي للبحث في مصفوفة غير مرتبة عن رقم طالب وطباعة اسم هذا الطالب. في الأسطر (١١-١٥) يتم البحث داخل المصفوفة id عن رقم الطالب الذي يتم إدخاله عن طريق لوحة المفاتيح (no)، فإذا كان هذا الرقم موجود في المصفوفة id يتم تخزين موقع رقم الطالب داخل المصفوفة id في المتغير index وبالتالي سوف تستخدم القيمة المخزنة في هذا المتغير لتحديد اسم الطالب صاحب هذا الرقم في المصفوفة name، حيث يتم من خلال الأسطر (١٦-١٩) إضافة اسم الطالب إلى المخرجات إذا كان رقم هذا الطالب موجوداً في المصفوفة id أو إضافة رسالة "There is no student with this ID !!!" إلى المخرجات إذا كان رقم الطالب المدخل غير موجود في هذه المصفوفة.

والشكل (١-١٦) يبين تنفيذ البرنامج السابق في حالة العثور على الطالب صاحب الرقم ٤.



شكل (١-١٦)

بينما يوضح الشكل (١-١٧) تنفيذ البرنامج السابق في حالة عدم العثور على أي طالب يحمل الرقم ١٣.



شكل (١-١٧)

مثال ١-١١:

```
// array11.java

import javax.swing.*;
class array11{
 public static void main(String args[]){
  int id[] = new int[] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  String  name[]=  {"Ahmad",  "Mohammed",  "Naser",  "Saleem",
                    "Adel",  "Khaled",  "Ali",  "Kareem",  "Jamil",
                    "Nader"};

  String s, stdName, title="The results of the Example (1-11)";
  String output="The student's name is:--> ";
  int no, index=-1;
  int low=0;
  int high = id.length-1;
  int middle;
  s=JOptionPane.showInputDialog("Enter the student's ID to display
                                   his name:");
  no=Integer.parseInt(s);

  while(low <=high){

 middle=(low+high)/2;
   if(no==id[middle]){
```

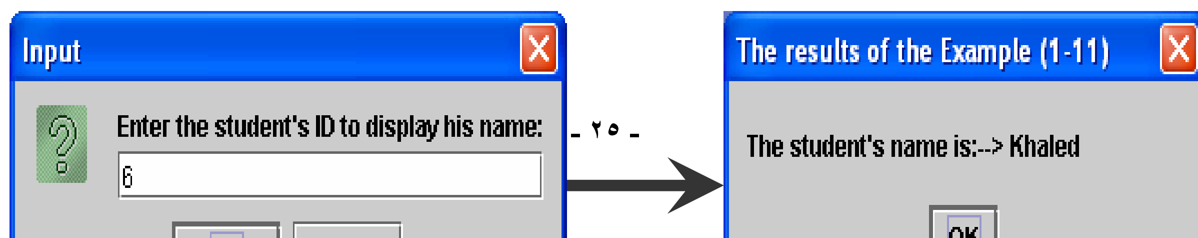
```
17.        index=middle;
18.        break;
19.       }
20.       else if(no<id[middle])
21.        high=middle-1;
22.       else
23.        low=middle+1;
24.      }

25.      if(index!=-1)
26.       output+=name[index];
27.      else
28.       output="There is no student with this ID !!!";
29.      JOptionPane.showMessageDialog(null, output, title,
                                                JOptionPane.PLAIN_MESSAGE)
30.      System.exit(0);
31.    }
32.  }
```

شرح المثال:

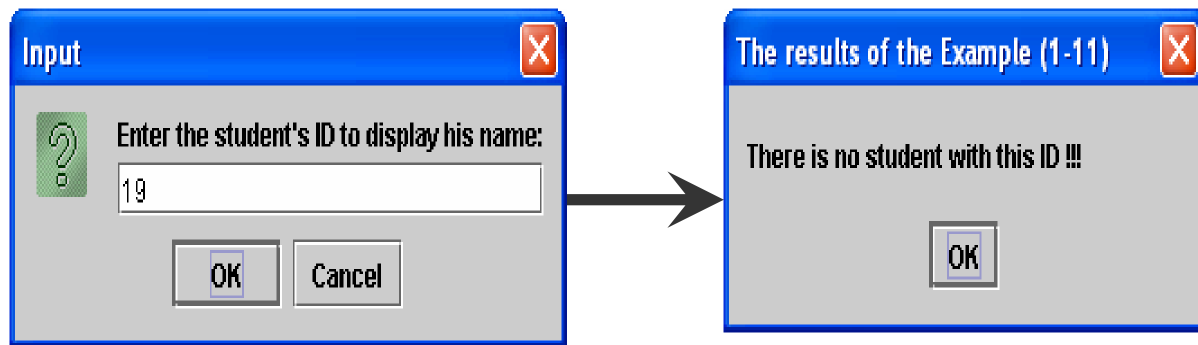
هذا المثال هو شبيه بالمثال السابق (١-١٠) حيث تم ترتيب أرقام الطلاب في المصفوفة id (السطر رقم ٤) واستخدمت طريقة البحث الثنائي والتي تعتبر أسرع وأكفأ عند البحث في المصفوفات المرتبة. في الأسطر (١٤-٢٤) تم تطبيق طريقة البحث الثنائي وذلك بتقسيم المصفوفة إلى نصفين وتحديد مكان وجود رقم الطالب المراد البحث عنه في أي نصف وبعد ذلك تقسيم النصف الذي ينتمي له رقم الطالب المراد البحث عنه إلى نصفين آخرين إلى أن يتم إيجاد رقم الطالب في المصفوفة id أو لغاية الخروج من الدوران (while) دون العثور على رقم الطالب في المصفوفة id.

والشكل (١-١٨) يبين تنفيذ البرنامج السابق في حالة العثور على الطالب صاحب الرقم ٦:



شكل (١-١٨)

بينما الشكل (١-١٩) يبين تنفيذ البرنامج السابق في حالة عدم العثور على طالب يحمل الرقم ١٩:



شكل (١-١٩)

مثال ١-١٢:

```
// array12.java

public class array9{
 public static void main(String[] args) {
  int[] testArray = new int[50];
  testArray[43] = 10;
 int testArray2[] = { 35, 23, 8, 34, 66, 88, 5, 2, 85, 33 };
  int key, index=-1;
  key=10;
  System.out.println("Searching for element == 10");

  for(int i = 0; i < testArray.length; i++) {
   if(testArray[i] == key)
   index = i;
```

```
12.     }

13.     if(index != -1)
14.      System.out.println("Element found at " + index);
15.     else
16.      System.out.println("Element (10) does not found at the array
                                  testArray");
17.     index=-1;
18.     key=88;
19.     System.out.println("Searching the second array for element ==
                                  88");

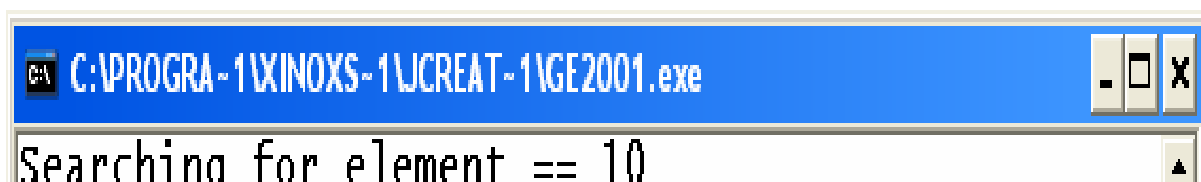
20.     for(int i = 0; i < testArray2.length; i++) {
21.      if(testArray2[i] == key)
22.      index = i;
23.     }

24.     if(index != -1)
25.      System.out.println("Element found at " + index);
26.     else
27.      System.out.println("Element (88) does not found at the array
                                  testArray2");
28.    }       }
```

شرح المثال:

وهذا المثال أيضاً يوضّح طريقة البحث الخطي، الأسطر (٩-١٢) والأسطر (٢٠-٢٣) توضّح عملية البحث الخطي عن طريق مقارنة العنصر المراد البحث عنه key مع جميع محتويات المصفوفة وبالترتيب، بحيث إذا وجد العنصر المراد البحث عنه تتم طباعة موقعه في المصفوفة وإذا لم يكن موجوداً في المصفوفة تتم طباعة رسالة بذلك.

والشكل (١-٢٠) يبين مخرجات البرنامج السابق.



شكل (١-٢٠)

المصفوفات ذات البعدين (Two-Dimensional Arrays):

في لغة جافا يمكن تعريف مصفوفات ذات أكثر من بعد واحد، وكمثال على ذلك: تعريف المصفوفات ذات البعدين. ونستطيع القول بأن المصفوفة ذات البعدين هي عبارة عن جدول يحتوي على صفوف وأعمدة، انظر الشكل (١-٢١). والمثال التالي يوضع كيفية تعريف مصفوفة ذات بعدين وحجز مواقع لها:

```
1. int b[][];
2. b = new int[ 3 ][ 4 ];
```

في السطر الأول تم تعريف مصفوفة ذات بعدين، وفي الصف الثاني تم حجز مواقع لهذه المصفوفة بحيث تحتوي على ثلاث صفوف كل صف منها يحتوي على ثلاثة أعمدة. والشكل (١-٢٠) يوضح المصفوفة b وأرقام مواقعها.

	العمود 0	العمود 1	العمود 2	العمود 3
الصف 0	b[0][3]	b[0][0]	b[0][1]	b[0][2]
الصف 1	b[1][0]	b[1][1]	b[1][2]	b[1][3]
الصف 2	b[2][0]	b[2][1]	b[2][2]	b[2][3]

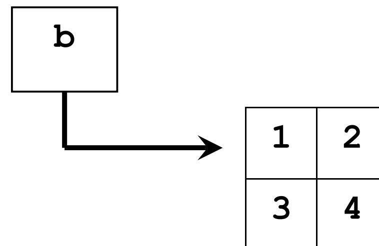
اسم المصفوفة
رقم الصـــف
رقم العمـــود

شكل (١-٢١)

والمثال التالي يوضح كيفية تعريف مصفوفة ذات بعدين وإعطائها قيماً ابتدائية:

```
int b[][] = { { 1, 2 }, { 3, 4 } };
```

في هذا المثال تم تخزين العدد ١ في المصفوفة b في الموقع الموجود في تقاطع الصف الأول والعمود الأول، والعدد ٢ في تقاطع الصف الأول والعمود الثاني. وبمعنى أخر تحتوي هذه المصفوفة على صفين، كل صف يحتوي على عنصرين. والشكل (١-٢٢) يبين محتويات المصفوفة b بعد تنفيذ الجملة السابقة:

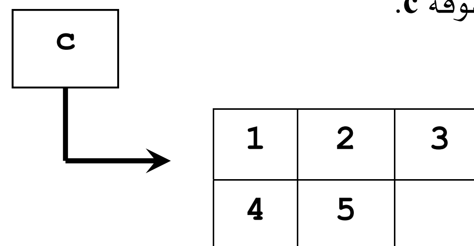


شكل (١-٢٢)

ويمكن لصفوف المصفوفة أن تحتوي على عدد مختلف من الأعمدة، بمعنى أن الصف الأول يحتوي على ثلاث أعمدة والصف الثاني يحتوي على عمودين فقط، كما في المثال التالي:

```
int c[][] = { { 1, 2, 3}, { 4, 5 } };
```

والشكل (١-٢٣) يبين محتويات المصفوفة **c**.

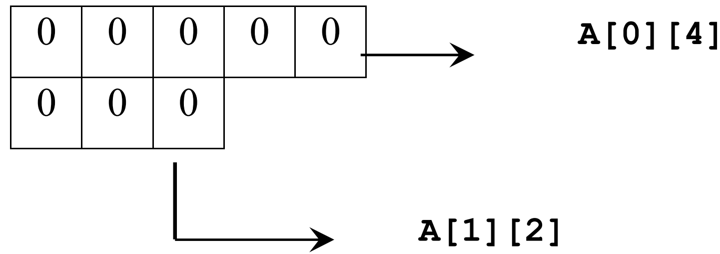


شكل (١-٢٣)

والمثال التالي يبين عملية تعريف مصفوفة وحجز مواقع لها بحيث يحتوي كل صف من صفوف هذه المصفوفة على عدد مختلف من الأعمدة:

```
1. int a[][];
2. a = new int[ 2 ][ ];   // allocate rows
3. a[ 0 ] = new int[ 5 ]; // allocate row 0
4. a[ 1 ] = new int[ 3 ]; // allocate row 1
```

في السطر رقم (١) تم تعريف مصفوفة اسمها a ، وفي السطر رقم (٢) تم حجز صفين لهذه المصفوفة ، بينما في السطر رقم (٣) تم حجز خمسة أعمدة للصف الأول، ومن خلال السطر رقم (٤) تم حجز ثلاث أعمدة للصف الثاني. والشكل (١-٢٤) يوضح المصفوفة a.



شكل (١-٢٤)

أمثلة على المصفوفات ذات البعدين:

مثال ١-١٣:

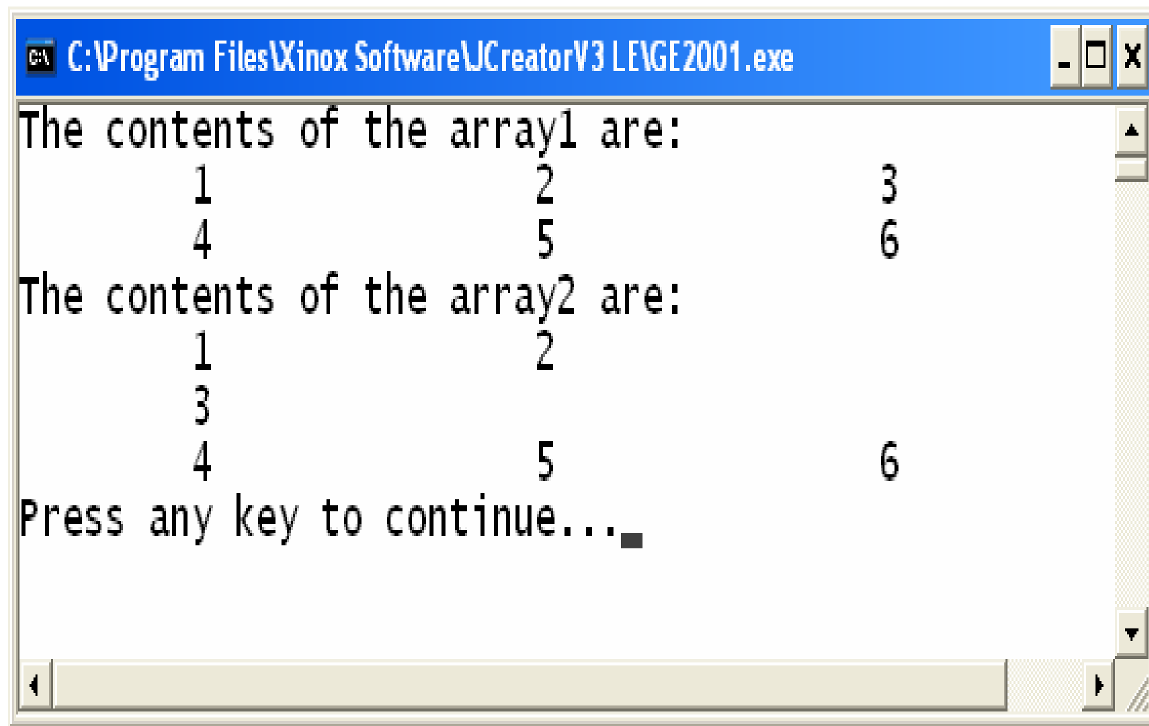
```
// array13.java

1.     public class array13 {
2.      public static void main(String[] args) {
3.       int array1[][] = { { 1, 2, 3 }, { 4, 5, 6 } };
4.       int array2[][] = { { 1, 2 }, { 3 }, { 4, 5, 6 } };
5.       System.out.println("The contents of the array1 are:");
6.       for(int i=0; i<array1.length; i++){
7.        for(int j=0; j<array1[i].length; j++)
8.         System.out.print("\t"+array1[i][j]+"\t");
9.        System.out.println();
10.      }
11.      System.out.println("The contents of the array2 are:");
12.      for(int i=0; i<array2.length; i++){
13.       for(int j=0; j<array2[i].length; j++)
14.        System.out.print("\t"+array2[i][j]+"\t");
15.       System.out.println();
16.      }
17.     }
18.    }
```

شرح المثال:

في السطر رقم (٣) تم تعريف مصفوفة اسمها array1 وتم إعطاؤها قيماً ابتدائية، بحيث احتوت هذه المصفوفة على صفين وكل صف احتوى على ثلاثة أعمدة. وفي السطر رقم (٤) تم تعريف مصفوفة اسمها array2 وتم إعطاؤها قيماً ابتدائية، بحيث احتوت على ثلاث صفوف، الصف الأول فيها احتوى على عمودين، والصف الثاني احتوى على عمود واحد فقط، بينما الصف الثالث احتوى على ثلاث أعمدة. ومن خلال الأسطر (٦-١٠) تم طباعة محتويات المصفوفة array1 بحيث تكون المخرجات على شكل جدول، وقد تم معرفة عدد الصفوف في المصفوفة array1 من خلال الجملة array1.length وتم معرفة عدد

الأعمدة في كل صف من خلال الجملة array1[i].length حيث i يمثل رقم الصف. وتم طباعة محتويات المصفوفة array2 من خلال الأسطر (١٢-١٦). والشكل (١-٢٥) يبين مخرجات هذا البرنامج.



شكل (١-٢٥)

مثال ١-١٤:

```
// array14.java

public class array14{

 public static void main(String[] args) {
  int grades[][] = { { 77, 68, 86, 73 },
                     { 96, 87, 89, 81 },
                     { 70, 90, 86, 81 } };

  int sum;
  System.out.println("The array is:");
```

```
8.      System.out.println("\t\t[0]\t[1]\t[2]\t[3]");

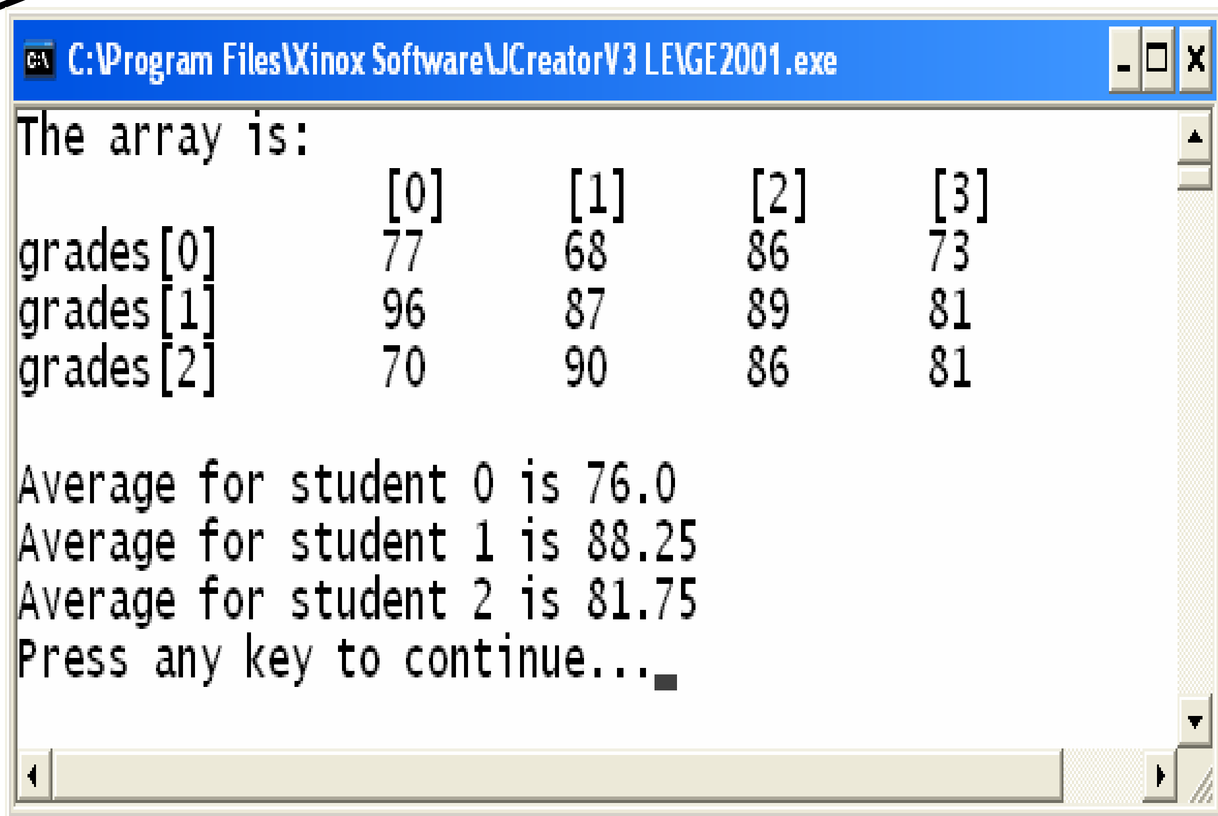
9.      for(int i=0; i<grades.length; i++){
10.      System.out.print("grades["+i+"]"+"\t");
11.      for(int j=0; j<grades[i].length; j++)
12.       System.out.print(grades[i][j]+"\t");
13.      System.out.println();
14.     }

15.     System.out.println();

16.     for(int i=0; i<grades.length; i++){
17.      sum=0;
18.      for(int j=0; j<grades[i].length; j++)
19.       sum+=grades[i][j];
20.      System.out.println("Average for student "+i+" is "
                                    +(double)sum/grades[i].length;
21.     }
22.    }
23.   }
```

شرح المثال:

هذا البرنامج يقوم بطباعة معدلات ثلاثة طلاب، كل طالب منهم له أربع درجات، حيث تم تعريف المصفوفة grades وتخزين الدرجات بها من خلال الأسطر (٣-٥). في الأسطر (٧-١٤) تم طباعة محتويات المصفوفة grades على شكل جدول مع توضيح أرقام الصفوف والأعمدة فيها. ومن خلال الأسطر (١٦-٢١) تم جمع درجات كل طالب لوحده (السطر رقم ١٩) ومن ثم إيجاد وطباعة المعدل لكل طالب (السطر رقم ٢٠)، حيث تم إيجاد المعدل لكل طالب وذلك بقسمة مجموع درجاته والمخزن في المتغير sum على عدد الدرجات والذي يمكن الحصول عليه لكل طالب من خلال الجملة grades[i].length حيث المتغير i يمثل رقم الصف (الطالب). يجب ملاحظة تصفير المتغير sum قبل جمع درجات كل طالب (السطر رقم ١٧). والشكل (١-٢٦) يبين مخرجات هذا البرنامج.



شكل (١-٢٦)

مثال ١-١٥:

```
// array15.java

public class array15{
 public static void main(String[] args) {
  int nums[][]= { {21, 24, 43, 54}, {15, 63, 27, 84}, {29, 10, 17, 42}, {28,
                                    33, 41, 67}
                                       };
  int sum=0;
  System.out.println("The contents of array nums are:");
  for(int i=0; i<nums.length; i++){
   for(int j=0; j<nums[i].length; j++)
    System.out.print(" "+nums[i][j]+" ");
   System.out.println();
  }
  for(int k=0; k<nums[1].length; k++)
```

```
12.      sum+=nums[1][k];
13.     System.out.println("\nThe sume of elements on the 2nd row is:
                                         "+sum);
14.     sum=0;
15.     for(int k=0; k<nums.length; k++)
16.      sum+=nums[k][2];
17.     System.out.println("\nThe sume of elements on the 3rd column is:
                                         "+sum);
18.     sum=0;
19.     for(int i=0; i<nums.length; i++)
20.      for(int j=0; j<nums[i].length; j++)
21.       if(i==j) sum+=nums[i][j];
22.        System.out.println("\nThe sume of elements on the main diagonal
                                             is: "+sum);
23.     sum=0;
24.     for(int i=0; i<nums.length; i++)
25.      for(int j=0; j<nums[i].length; j++)
26.       if(i+j==nums.length-1) sum+=nums[i][j];
27.        System.out.println("\nThe sume of elements on the secondary
                                            diagonal is: "+sum);
28.     System.out.println();
29.    }
30.   }
```

شرح المثال:

في هذا المثال تم التعامل مع صفوف وأعمدة معينه داخل المصفوفة، وتم التعامل مع القطر الرئيسي (ويمكن تحديده عندما يكون رقم الصف يساوي رقم العمود) والقطر الثانوي (ويمكن تحديده عندما يكون مجموع رقم الصف مع رقم العمود ناقص واحد يساوي عدد الصفوف (أو عدد الأعمدة) للمصفوفة، يجب ملاحظة أن القطر الرئيسي والقطر الثانوي يمكن التعامل معهم فقط في المصفوفات المربعة (المصفوفات المربعة هي المصفوفات التي يكون فيها عدد الصفوف وعدد الأعمدة متساويين). في الأسطر (٧-١١) تم طباعة محتويات المصفوفة على شكل جدول. في الأسطر (١٢-١٤) تم إيجاد وطباعة مجموع الأرقام المخزنة في الصف الثاني (رقم هذا الصف في المصفوفة هو ١). بينما تم في الأسطر (١٦-١٨) تم إيجاد وطباعة مجموع الأرقام المخزنة في العمود الثالث (رقم هذا العمود في المصفوفة هو ٢). وتم إيجاد وطباعة مجموع الأرقام الموجودة على القطر الرئيسي للمصفوفة nums من خلال الأسطر

(٢٠-٢٣)، لاحظ أنه باستخدام الشرط في السطر رقم (٢٢) تم معرفة فيما إذا كان هذا العنصر موجود على القطر الرئيسي أما لا. ومن خلال الأسطر (٢٥-٢٨) تم إيجاد وطباعة مجموع الأرقام الموجودة على القطر الثانوي للمصفوفة، لاحظ الشرط الذي من خلاله تم تحديد فيما إذا كان العنصر موجود على القطر الثانوي أم لا وذلك في السطر رقم (٢٧). والشكل (١-٢٧) يبين مخرجات هذا البرنامج.

```
C:\Program Files\Xinox Software\JCreatorV3 LE\GE2001.exe

The contents of array nums are:
 21  24  43  54
 15  63  27  84
 29  10  17  42
 28  33  41  67

The sume of elements on the 2nd row is: 189

The sume of elements on the 3rd column is: 128

The sume of elements on the main diagonal is: 168

The sume of elements on the secondary diagonal is: 119

Press any key to continue...
```

شكل (١-٢٧)

تماريـــن:

س١: شركة تمنح موظفيها راتباً شهرياً مقداره ٢٥٠٠ ريال سعودي، وتمنح الشركة نسبة ٩٪ من مبيعات الموظف كعمولة تضاف إلى راتبه الشهري. اكتب برنامج يقرأ رواتب ١٠ موظفين ومجموع مبيعاتهم الشهرية بحيث تكون مخرجات البرنامج عبارة عن جدول يحتوي على عمولة الموظف وإجمالي راتب الموظف المكتسب في نهاية الشهر (إجمالي الراتب هو راتب الموظف ٢٥٠٠ ريال سعودي + ٩٪ من مجموع مبيعات الموظف في ذلك الشهر).

س٢: اكتب برنامج لقراءة ٢٠ عدد صحيح وتخزينها في مصفوفة ومن ثم فحص جميع الإعداد المخزنة في هذه المصفوفة وتخزين الأعداد الفردية في مصفوفة أخرى. وفي نهاية البرنامج اطبع محتويات المصفوفتين. (ملاحظة: يجب أن يكون حجم المصفوفة التي ستحوي الأعداد الفردية مساوياً لعدد هذه الأعداد).

س٣: اكتب برنامج لقراءة معدلات وأسماء ١٠ طلاب وتخزينهم في مصفوفتين (مصفوفة للمعدلات ومصفوفة للأسماء)، بحيث يقوم البرنامج بطباعة أسماء الطلاب الناجحين (اللذين تزيد معدلاتهم عن أو تساوي ٦٠) واسم الطالب صاحب أعلى درجة.

س٤: اكتب برنامج لقراءة N من الأعداد الحقيقة وتخزينها في مصفوفة، بحيث يقوم البرنامج بترتيب محتويات المصفوفة ترتيباً تصاعدياً. ويقوم البرنامج بقراءة عدد حقيقي من لوحة المفاتيح ليقوم بالبحث عن هذا العدد في المصفوفة بطريقة البحث الثنائي، فإذا وجد هذا العدد في المصفوفة يطبع البرنامج مكان وجود هذا العدد في المصفوفة وفي حال عدم وجوده يطبع "Not found in the array".

س٥: اكتب برنامج لتخزين أرقام وأسماء ورواتب موظفين في ثلاث مصفوفات. بحيث يستطيع المستخدم لهذا البرنامج البحث عن اسم وراتب موظف معين عن طريق رقمه (استخدم طريقة البحث الخطي). وكذلك يقوم البرنامج بطباعة أرقام وأسماء الموظفين اللذين تزيد رواتبهم عن ٢٥٠٠ ريال سعودي.

س٦: اكتب برنامج لتخزين جدول ضرب الخمسة في مصفوفة ذات بعدين، ومن ثم يقوم البرنامج بطباعة محتويات هذه المصفوفة.

س٧: لديك المصفوفة التالية:

٤	٦	٤	٩	٢
١	٣	٩	١	٦
٩	٩	٢	٥	٦
٣	٧	٤	٣	٨
٥	٣	٥	٦	٣

اكتب برنامج لطباعة ما يلي:

- مجموع الأعداد المخزنة في الصف الثاني والصف الرابع.
- الأعداد المخزنة في العمود الثالث.
- مجموع الأعداد المخزنة في القطر الرئيسي.
- معدل الأعداد المخزنة في القطر الثانوي.



برمجة ٢

الطـــــرق

الطــرق

٢

الجــــدارة:

معرفة كيفية كتابة الطرق بجميع أشكالها، ومعرفة كيفية استخدام الطرق الخاصة بالسلاسل الرمزية (String). بالإضافة للتعامل مع الطرق الموجودة في الصنف (Math).

الأهــــداف:

عندما تكمل هذه الوحدة تكون قادراً على:

١- فهم ماهية الطرق (تعريفها واستخدامها).
٢- استخدام الطرق الموجودة مع الصنف (Math Class).
٣- معرفة فترة الحياة للمتغيرات (Life Time).
٤- معرفة مجال المتغيرات (Scope).
٥- معرفة استخدام الاستدعاء الذاتي (Recursion).
٦- معرفة واستخدام مفهوم (Overloading).
٧- التعامل مع الطرق الخاصة بالسلاسل الرمزية (String).

مستوى الأداء المطلوب:

أن يصل المتدرب إلى إتقان هذه الجدارة بنسبة ١٠٠٪.

الوقت المتوقع للتدريب: ١٠ ساعات.

الوسائل المساعـدة:

- قلم.
- دفتر.
- جهاز حاسب آلي.

متطلبات الجدارة:

اجتياز جميع الحقائب السابقة.

مقدمة :

في هذا الفصل سنقوم بالتعرف على كيفية تعريف الطرق وكيفية التعامل معها وأشكال استدعائها. وكذلك سنقوم باستخدام الطرق المتوفرة في بعض الأصناف الجاهزة والمتوفرة في مكتبة جافا ، مثل تلك الطرق الخاصة بالتعامل مع السلاسل الرمزية والطرق الخاصة بالعمليات الحسابية ، وفي نهاية هذه الوحدة هنالك عدد من التمارين.

ما هي الطرق (Methods)؟

الطريقة هي عبارة عن مجموعة من الجمل وتعرف بجسم الطريقة (Method Body) حيث يكون لها اسم معين، وتعرّف داخل الصنف. وتعرف الطريقة من خلال التوقيع (Signature) الخاص بها ، وهو عبارة عن اسم الطريقة ، نوع المعاملات وترتيبها ، بالإضافة إلى نوع البيانات الراجعة منها.

والآن لنتعرف على عملية استخدام الطرق وذلك باستخدام تلك الطرق الموجودة في صنف العمليات الحسابية (Math Class).

صنف العمليات الحسابية (Math Class):

يحتوي هذا الصنف على العديد من الطرق التي تقوم بالعمليات الحسابية الشائعة مثل إيجاد القيمة المطلقة لعدد ،قوة العدد . . . الخ. وتتم عملية استدعاء الطرق بكتابة اسم الصنف متبوعا بنقطة بعدها اسم الطريقة ثم قائمة المعاملات داخل أقواس دائرية ، كما يلي:

Class_Name.method_Name(*Argument List*)

مثال:

```
System.out.println(Math.sqrt(9.0)) ;
```

تقوم هذه الجملة باستدعاء الطريقة (sqrt) الموجودة في الصنف (Math) والتي تأخذ معامل واحد (9.0) من نوع (Double). فنتيجة تنفيذ هذه الجملة ستكون طباعة 3.0. والجدول (٢-١) يحتوي بعض الطرق الموجودة في الصنف (Math).

الطريقة	وصف الطريقة	مثال
`abs(x)`	القيمة المطلقة لـ **x**.	`Math.abs(6.2) → 6.2` `Math.abs(-2.4)→ 2.4`
`ceil(x)`	تقرّب **x** إلى أقل عدد صحيح ليس اقل من **x**.	`Math.ceil(5.1) → 6` `Math.ceil(-5.1)→ -5`
`floor(x)`	تقرب **x** إلى أكبر عدد صحيح ليس أكبر من **x**.	`Math.floor(5.1) → 5` `Math.floor(-5.1)→-6`
`max(x,y)`	أكبر قيمة من **x** و **y**.	`Math.max(7,6) → 7`
`min(x,y)`	أقل قيمة من **x** و **y**.	`Math.min(-7,-8) → -8`
`pow(x,y)`	**x** مرفوعة للأس **y**.	`Math.pow(6,2)→`6^2` → 36`
`sqrt(x)`	الجذر التربيعي لـ **x**.	`Math.sqrt(9)→` $\sqrt{9}$ `→ 3`
`random()`	تكوّن رقم عشوائي بين الصفر والواحد.	`Math.random()→ 0.23121`

جدول (٢-١)

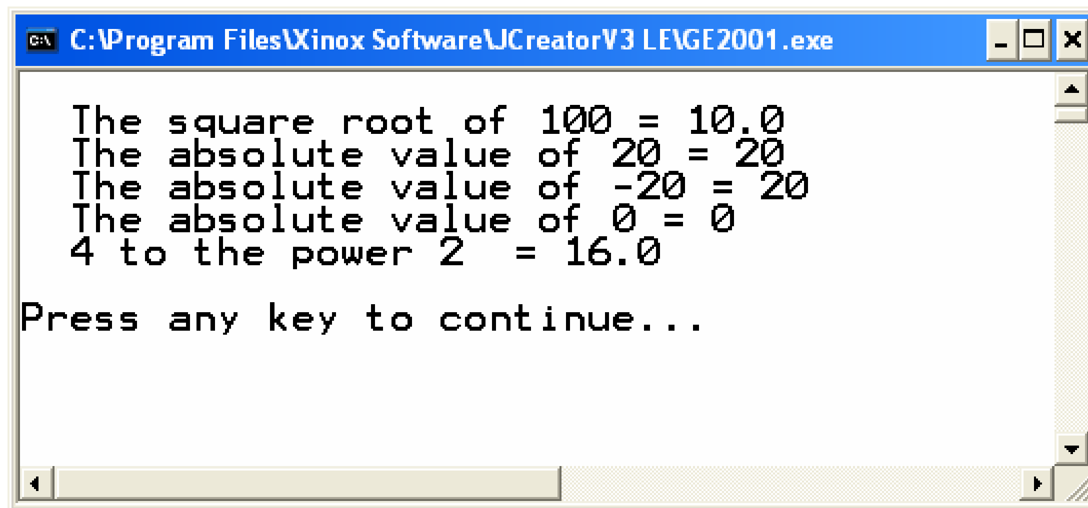
مثال ٢-١:

```
// UseMath.java

public class UseMath{
 public static void main( String args[]){
  System.out.println("The square root of 100 = " + Math.sqrt(100));
  System.out.println("The absolute value of 20 = " + Math.abs(20));
  System.out.println("The absolute value of -20 = " + Math.abs(-20));
  System.out.println("The absolute value of 0 = " + Math.abs(0));
  System.out.println("4 to the power 2  = " + Math.pow(4,2));
 }     // end of main
}      // end of class UseMath
```

شرح المثال:

من خلال المثال (٢-١) قمنا بالتعامل مع بعض الطرق الموجودة مع الصنف (Math)، والشكل (٢ - ١) يبين مخرجات هذا البرنامج.



شكل (٢-١)

وهنالك الكثير من المسائل التي تحتاج إلى استخدام الأرقام العشوائية مثل الألعاب وبرامج المحاكاة والمسابقات وغيرها. سنتعرف في المثال (٢-٢) على كيفية توليد الأرقام العشوائية واستخدامها من خلال مثال رمي حجر نرد ٥ مرات .

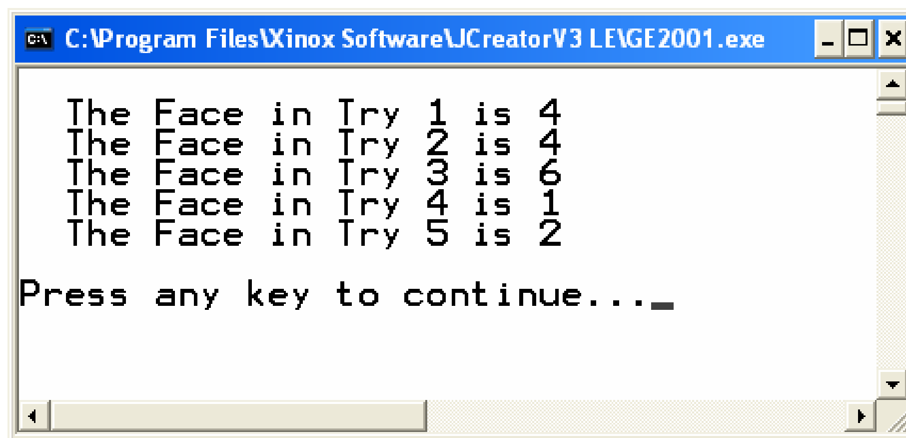
مثال (٢-٢):

```
// RollDie.java

public class RollDie{
 public static void main( String args[]){
  int face ;//variables to store the result
  for (int i = 1;i<=5;i++){
   face = 1+(int)(Math.random()*6);
 System.out.println("The Face in Try " + i + " is " + face);
   } // end for loop
  } // end of main
} // end of class RollDie
```

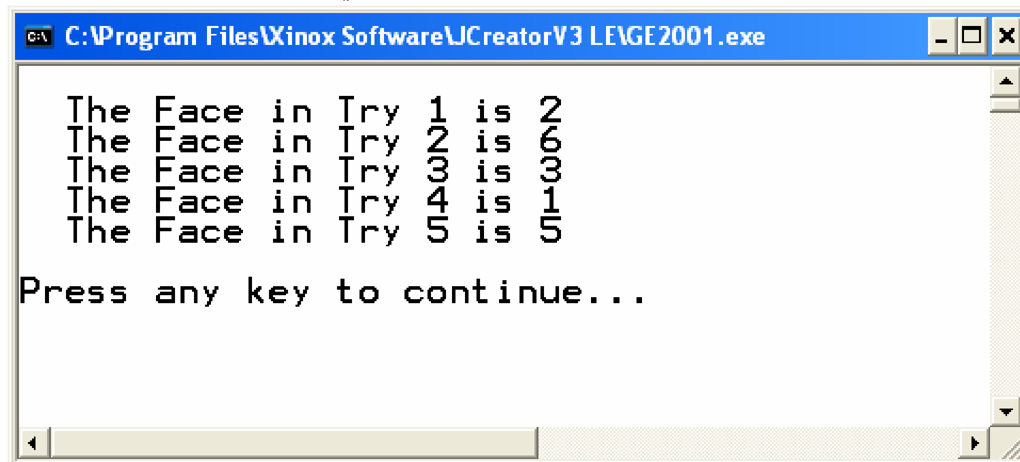
شرح المثال:

في هذا المثال يقوم البرنامج بتوليد ٥ أرقام عشوائية وذلك من خلال تنفيذ الجملة رقم (٥)، حيث تم استدعاء الطريقة random الموجودة في الصنف Math، وتقوم هذه الطريقة بتوليد رقم عشوائي أكبر من أو يساوي الصفر وأقل من واحد، ومن ثم تم ضرب الرقم العشوائي الناتج من (Math.random()) بالرقم ٦ وتحويله إلى عدد صحيح من خلال (int) ليصبح العدد العشوائي الناتج أكبر من أو يساوي صفر وأقل من أو يساوي خمسة، ومن ثم يضيف إلى الرقم العشوائي الرقم ١ ليصبح الرقم الناتج من تنفيذ السطر رقم (٦) اكبر من أو يساوي واحد وأقل من أو يساوي ستة. والشكل (٢-٢) يبين مخرجات هذا البرنامج. لاحظ أن نتائج هذا البرنامج قد تختلف في كل مرة ننفذ فيها البرنامج.



شكل (٢-٢)

والشكل (٢-٣) يبين مخرجات البرنامج بعد تنفيذه مرة أخرى، وهي نتائج مختلفة كما تلاحظ.



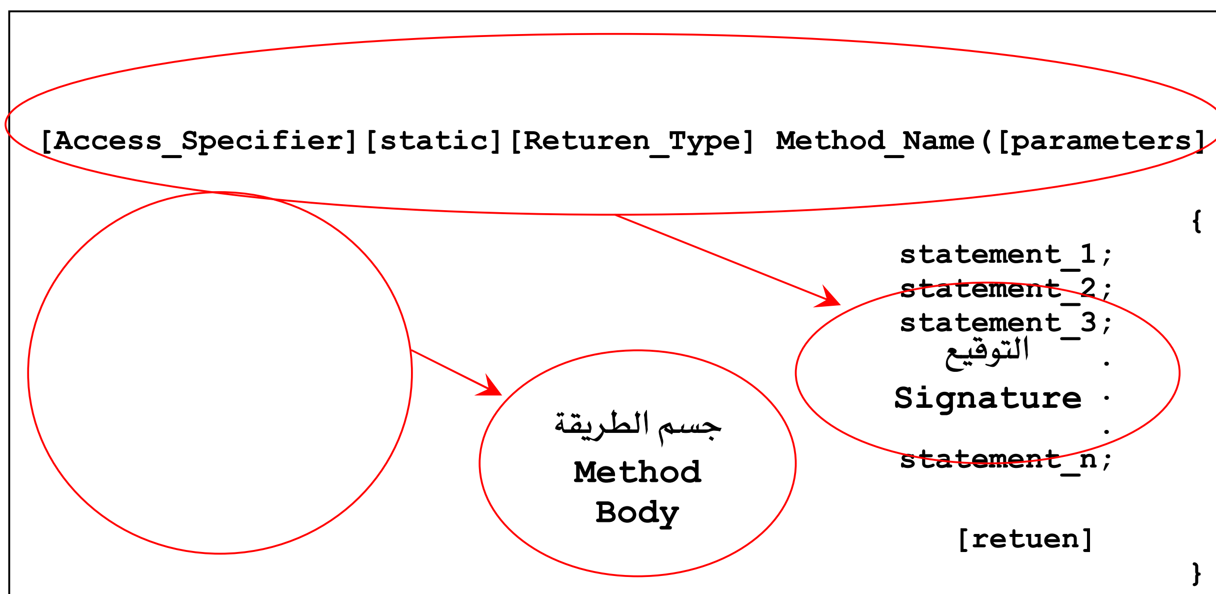
شكل (٢-٣)

فوائد استخدام الطرق:

ثبت عملياً أن أفضل طريقة لحل المسائل هي تقسيم هذه المسائل إلى وحدات صغيرة ويعرف هذا المبدأ بمبدأ "فرّق تسد" (Divide and Conquer) والطرق في جافا توفر لنا الإمكانية الكافية لتطبيق هذا المبدأ ، مما سيسهل علينا كتابة البرنامج وتتبعه وإمكانية فهمه وصيانته بسهولة. وكذلك فإن استخدام الطرق المعرفة سابقا يوفر علينا كتابة البرنامج، وذلك من خلال إعادة استخدام هذه الطرق دون الحاجة إلى كتابتها مرة أخرى ودون الحاجة إلى معرفة ماهية هذه الطرق وكيف كتبت (Software Reusability)، ومثال ذلك استخدام تلك الطرق الموجودة في صنف العمليات الحسابية (Math Class)، والفائدة الأخرى هي تفادي تكرار كتابة الجمل في البرنامج، فما علينا سوى كتابة الجمل التي نحتاج إلى تكرارها في البرنامج داخل طريقة (Method)، ومن ثم نقوم باستدعاء هذه الطريقة عن طريق اسمها في أكثر من موقع في البرنامج ، كما سنرى لاحقاً.

تعريف الطرق واستدعائها:

كما ذكرنا سابقاً فالطريقة هي عبارة عن مجموعة من الجمل (وتعرف بجسم الطريقة (Method Body) حيث يكون لها اسم معين، وتعرّف داخل الصنف. وتعرّف الطريقة من خلال التوقيع (Signature) الخاص بها، وهو عبارة عن اسم الطريقة، نوع المعاملات وترتيبها، بالإضافة إلى نوع البيانات الراجعة منها، والشكل (٢-٤) يبين الشكل العام لتعريف الطرق.



شكل (٢-٤)

وفي ما يلي شرح الشكل العام لتعريف الطرق، الأقواس المربعة "[" و "]" تدل على أن المحصور بينهما هو اختياري، أي يمكن أن يحذف من التوقيع الخاص للطريقة، لكن عند حذفها يجب مراعاة أمور أخرى سنتطرق لها لاحقاً في هذه الحقيبة إن شاء الله.

-(Access_Specifier) وهو محدد الوصول، ويمكن أن يكون واحد من المحددات التالية:

-(private): أي بمعنى "خاص"، بحيث تكون الطريقة (Method) مرئية فقط داخل الصنف (Class) الذي عرّفت فيه.

-(public): أي بمعنى "عام"، وتكون الطريقة مرئية في أي مكان في البرنامج.

-إذا لم يتم كتابة محدد الوصول (Access_Specifier) هذا يدل على أن هذه الطريقة مرئية في داخل الحزمة التي يتبع لها الصنف الذي عرفت الطريقة فيه.

وهنالك محددات وصول أخرى سوف يتم التطرق لها لاحقاً في الوحدة الثالثة من هذه الحقيبة.

-(static) اي بمعنى "ثابت"، وتستخدم لتعريف الطرق ليتم استخدامها داخل الصنف الذي عرّفت فيه فقط. (أي لا يمكن أن ترتبط بأي كائن (Object) من نوع هذا الصنف). وسوف نتطرق للكائن في الوحدة الثالثة، إن شاء الله.

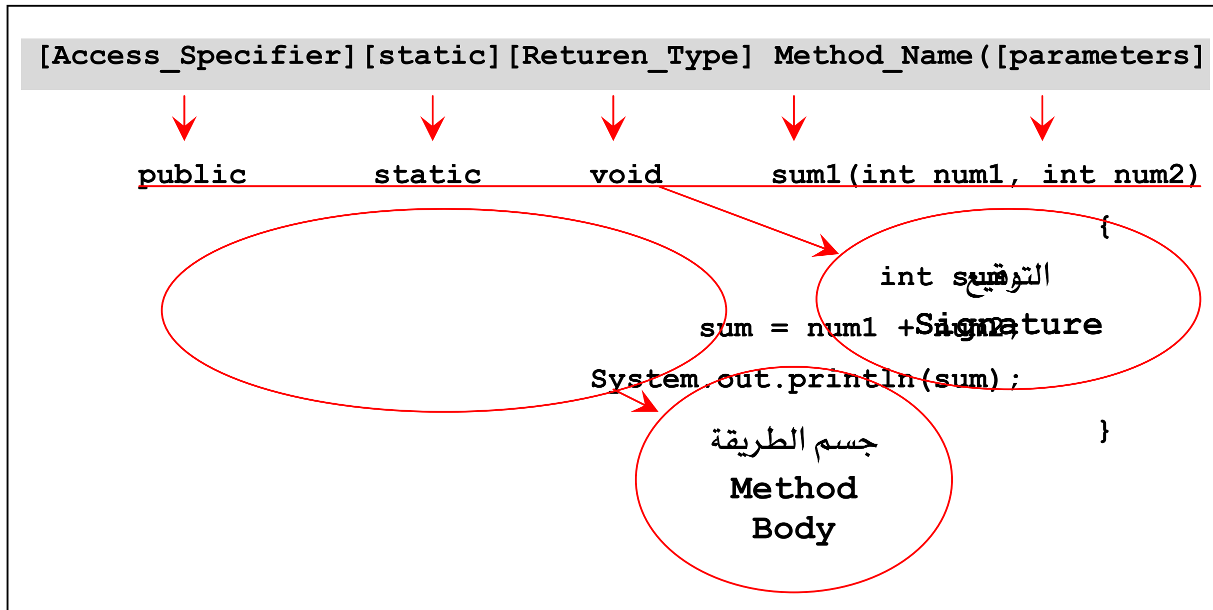
-(Return_Type) وهذا يحدد نوع البيانات التي ترجعها الطريقة عند استدعائها، ويمكن أن يكون نوع البيانات المرجعة أي نوع من أنواع البيانات (Data Types) التي تعرفها من خلال دراستك لمادة برمجة-١ (مثل:int، char، . . . الخ)، ويتم إرجاع القيمة باستخدام الكلمة المحجوزة (Return). ويمكن للطريقة أن لا ترجع أي قيمة، وفي هذه الحالة يجب أن يكون نوع البيانات المرجعة void.

-(Method_Name) وهو اسم الطريقة، ويجب مراعاة الشروط الخاصة بتحديد أسماء المتغيرات عند اختيار اسم للطريقة.

-(parameters) وهي المعاملات، وعند تعريف الطريقة تسمى هذه المعاملات بالمعاملات الشكلية (Formal Parameters)، ويمكن أن تستخدم هذه المعاملات في جسم الطريقة كمتغيرات بالإضافة للمتغيرات المحلية (Local Variables) التي تعرّف داخل جسم الطريقة. وعند استدعاء الطريقة تسمى المعاملات بالمعاملات الفعلية(Actual Parameters).

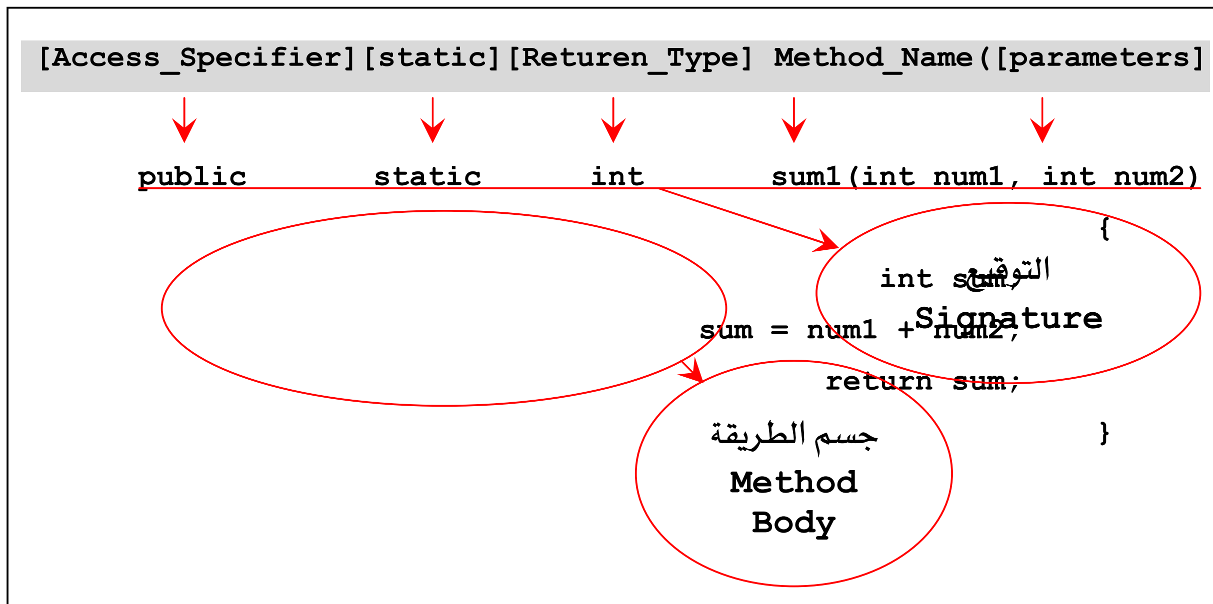
-(Method_Body) وهو جسم الطريقة، ويمكن أن يحتوي على تعريف المتغيرات المحلية والجمل التي يتم تنفيذها عند استدعاء هذه الطريقة. وإذا كان نوع البيانات المرجعة من هذه الطريقة غير النوع void فيجب أن يحتوي جسم الطريقة على جملة return التي توقف عمل الطريقة وترجع القيمة منها إلى مكان الاستدعاء.

والشكل (٢-٥) يبين كيفية تعريف طريقة لا ترجع أي قيمة (من نوع void)، بحيث تقوم هذه الطريقة والتي أسمها sum1 بجمع عددين وطباعة مجموعهما.



شكل (٢-٥)

والشكل (٢-٦) يبين كيفية كتابة الطريقة السابقة sum1 بحيث ترجع قيمة صحيحة وهي ناتج جمع العددين.



شكل (٢-٦)

ملاحظة: لا يجوز في لغة جافا كتابة طريقة داخل طريقة أخرى اطلاقاً.

مثال (٢-٣):

```
// Methods.java

1.  public class Methods {
2.  // instance variable declaration . . .
3.  public void method1(){
4.      //body
5.  }
6.  public void method2(int i , double j){
7.      //body
8.  }
9.
10. public int method3(){
11.         //body
12. return 0; //integer expression
13. }
14.
15. public int method4(int i ,String s ){
16.         //body
17. return  0; //integer expression
18. }
19.
20. }
```

شرح المثال:

في المثال (٢-٣) تم تعريف أربع طرق لتوضيح الأشكال التي يمكن للطرق أن تأتي بها. خلال الاسطر (٣-٥) تم تعريف الطريقة (Method1) لا تأخذ معاملات ولا ترجع قيمة. والأسطر (٦-٨) تعرّف طريقة اسمها (Method2) لا ترجع قيمة لكن تأخذ المعاملات التالية: i من نوع int و j من نوع double. وفي الأسطر (١٠-١٣) تم تعريف الطريقة (Method3) والتي لا تأخذ معاملات لكن ترجع قيمة من نوع (int).

بينما في الاسطر (١٥-١٨) عرفت الطريقة (Method4) والتي تأخذ المعاملات التالية: i من نوع int و s من نوع String. وترجع قيمة من نوع (int).

تتم عملية استدعاء الطرق وبكل بساطة عن طريق كتابة اسم الطريقة وإرسال قيم المعاملات إن وجدت. ويتم ذلك في المكان المراد تنفيذ عمل الطريقة فيه، ويجب أن يكون الاستدعاء داخل طريقة أخرى. والشكل (٢-٧) يبين الشكل العام لعملية استدعاء الطرق.

```
Method_Name ( [Parameters_List] );
```

شكل (٢-٧)

وفي مايلي شرح الشكل العام لعملية استدعاء الطرق:

-(Method_Name): اسم الطريقة، وعند استدعاء طريقة موجودة في صنف آخر لابد من كتابة اسم هذا الصنف قبل اسم الطريقة بحيث تفصل بينهم نقطة.

-(Parameters_List): قائمة المعاملات الفعلية (Actual Parameters)، وهي القيم الفعلية التي تستخدم في عملية استدعاء الطرق، ويمكن أن تكون بالأشكال التالية:

- قيم ثابتة، مثل: sum1(5, 6).
- متغيرات، مثل: sum1(x, y).
- استدعاء لطريقة (Method) أخرى، مثل: sum1(sum2(z, 4), y).

وتأتي عملية استدعاء الطرق على شكلين هما:

١ -الطرق التي لا ترجع قيم (void) وفي هذه الحالة يجب أن لا يتم إسنادها إلى متغير ولا استخدامها في تعبير.

٢ -الطرق التي تقوم بإرجاع قيم وفي هذه الحالة يجب أن تستخدم في إحدى الحالات التالية :

- يتم إسنادها إلى متغير.
- استخدامها في تعبير.
- استخدامها في عملية استدعاء لطريقة أخرى، مثل: إرسالها للطريقة الخاصة بالطباعة System.out.println( ).

مثال (٢-٤):

```
// MethodCall.java

1.  public class MethodCall {
2.  public  static   void main(String args[]){
3.   int x = 5, y = 6, z = 0, s = 0;
4.   sum1(10, 5);
5.   sum1(x, y);
6.   s  =sum2(5, 6);
7.   System.out.println("sum = " +  sum2(5, 6));
8.   z = 12 + 3 * sum2(x, 10);
9.   sum1(sum2(3, 4), 5);
10. } // end of  main
11.
12. // defining the method sum1
13. static void sum1(int num1,int num2){
14.   int sum=0;//local variable
15.   sum= num1+num2 ;
16.   System.out.println("sum = "+ sum);
17. } // end of sum1
18.
19. // defining the method sum2
20. static int sum2(int num1,int num2){
21.   int sum=0; //  local variable
22.   sum= num1+num2 ;
23.   return sum ;// returned value
24.  }        // end of  sum2
25. }// end of class MethodCall
```

شرح المثال:

في هذا المثال سوف نقوم بتوضيح عمليات الاستدعاء بشكليها السابقين. في السطر (٤) والسطر (٥) تم استدعاء الطريقة sum1 دون اسناد هذه الطريقة إلى أي متغير وذلك لأن هذه الطريقة لا ترجع أي قيمة

ونوع القيمة المرجعة فيها هو void، بحيث تم إرسال ثوابت في الاستدعاء الأول وتم إرسال متغيرات في الاستدعاء الثاني. بينما الطريقة sum2 ترجع قيمة من نوع int، في السطر (٦) تم إرسال ثوابت للطريقة sum2،وتم إسناد استدعاء هذه الطريقة إلى المتغير s ليتم تخزين القيمة المرجعة من هذه الطريقة في هذه المتغير. وفي السطر (٧) تم إرسال استدعاء الطريقة sum2 إلى الطريقة الخاصة بالطباعة وهي System.out.println(). والسطر (٨) يبين كيفية استدعاء الطريقة sum2 داخل تعبير حسابي. السطر (٩) يوضح عملية إرسال استدعاء الطريقة sum2 إلى الطريقة sum1.

والشكل (٢-٨) يبين نتائج تنفيذ البرنامج في المثال (٢-٤) السابق.

```
C:\Program Files\Xinox Software\JCreatorV3 LE\GE2001.exe

 sum = 15
 sum = 11
 sum = 11
 sum = 12

Press any key to continue...
```

شكل (٢-٨)

ماذا يحدث عند استدعاء الطريقة (method)؟

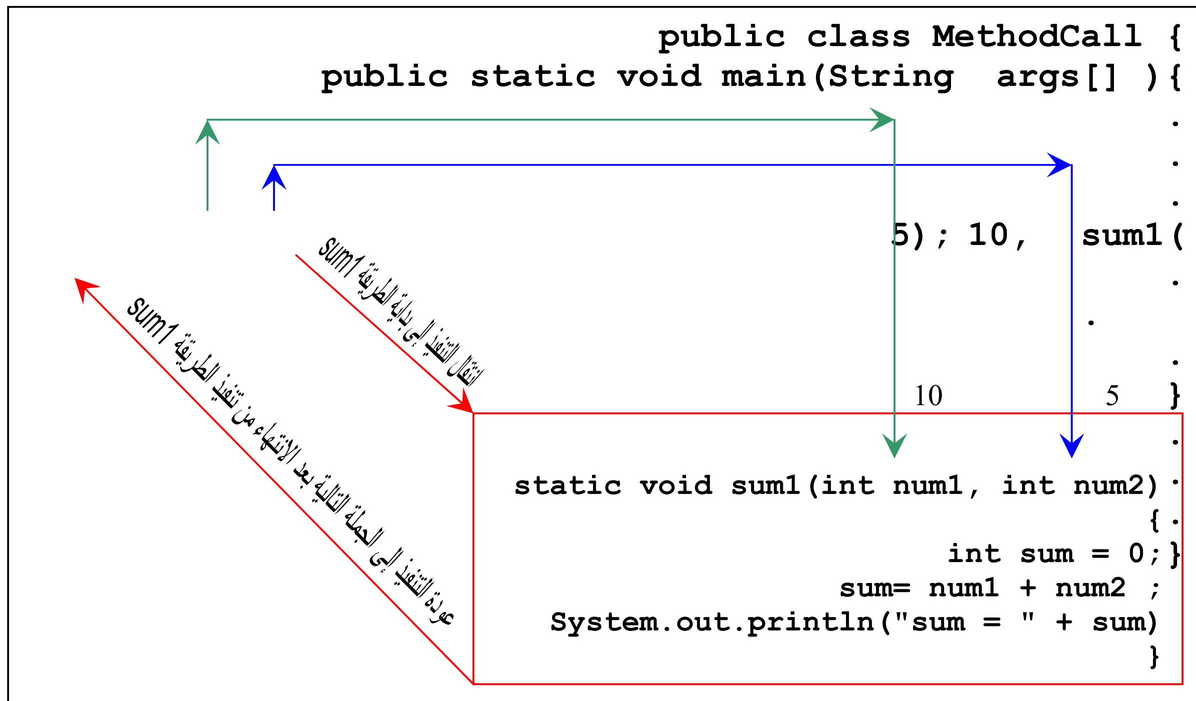
١- تنسخ المعاملات الفعلية (Actual parameters) إلى المعاملات الشكلية (parameters Formal) أي تصبح المعاملات الفعلية كقيم ابتدائية للمعاملات الشكلية. وتعمل المعاملات الشكلية عمل المتغيرات المحلية في جسم الطريقة.

٢- ينتقل تنفيذ البرنامج إلى بداية الطريقة المستدعاة .

٣- عند الانتهاء من تنفيذ الطريقة يستمر تنفيذ البرنامج من الجملة التالية لجملة الاستدعاء.

وكمثال على هذه الخطوات نعود إلى مثال (٢-٤) فمثلا عند استدعاء sum1(10,5) تنسخ القيمة ١٠ لِ num1، والقيمة ٥ لِ num2 ثم بعد ذلك ينتقل التنفيذ إلى بداية الطريقة sum1 ويستمر التنفيذ حتى نهاية

هذه الطريقة أو عند مواجهة جملة الرجوع (return). ثم بعد ذلك يعود التنفيذ إلى أول جملة بعد جملة الاستدعاء لِ sum1 وهي الجملية الموجودة في السطر (٥). والشكل (٢-٩) يبين خطوات بدء تنفيذ الطريقة sum1 في السطر (٤) والعودة منها بعد الانتهاء من التنفيذ إلى السطر (٥).



شكل (٢-٩)

فترة حياة المتغيرات (Variable Life Time) :

فترة الحياة للمتغير: هي الفترة التي يبقى فيها المتغير موجوداً داخل الذاكرة العشوائية (RAM)خلال تنفيذ البرنامج.

كما مر معنا سابقا فقد واجهنا أربعة أنواع من المتغيرات، فلنستعرض هذه الأنواع الأربعة من المتغيرات مع فترة حياتها بالنسبة للبرنامج :

١. المتغيرات الثابتة Static Variables: هي المتغيرات الخاصة بصنف أي أنها غير مرتبطة بأي كائن ينتمي لهذا الصنف. وتبدأ فترة حياة هذه المتغيرات عند عملية التحميل للصنف وتنهي عند إعادة التحميل لهذا الصنف.

٢. المتغيرات المحلية Local Variables: وهي المتغيرات المعرفة على مستوى المقطع (Block) الذي عرفت بداخله. أما فترة الحياة للمتغيرات المحلية فتبدأ عند إنشاء هذه المتغيرات وتنتهي عند الخروج من المقطع (block) الذي عرف به المتغير.

٣. المعاملات Parameter Variables: وهي التي تم تعرفيها في تعريف الطريقة (Method). وبالنسبة للمعاملات فتبدأ فترة الحياة عند استدعاء الطريقة (Method) وتنتهي عند الرجوع منها.

٤. متغيرات المثال Instance Variables: وهي المتغيرات الخاصة بالمثال (النسخة) المنشئ من صنف معين (وسوف تتم دراستها بالتفصيل في الفصل الخاص بالأصناف). وتبدأ فترة حياة متغيرات المثال عند إنشاء الكائن وتبقى مادامت أجزاء البرنامج تستطيع الوصول إلى هذا الكائن.

مجال المتغيرات (Variable Scope):

وهو الجزء من البرنامج الذي نستطيع من خلاله الوصول إلى المتغير. فبالنسبة لمتغيرات النسخة (Instance Variables) والطرق (Method) فنستطيع الوصول إليها داخل الصنف أي من بداية تعريف الصنف وحتى نهاية تعريفه. أما المتغيرات المحلية فإمكانية الوصول إليها تكون داخل المقطع (Block) الذي عرفت به فقط. اما بالنسبة للمتغيرات المحلية المعرفة على مستوى الطريقة والمعاملات فتكون إمكانية الوصول إليها داخل تلك الطريقة فقط .

مثال (٢-٥):

```
// VariableScope.java

public class VariableScope{
 static int i;              //instance variable
 public static void main(String args[]){
  int x = 5, y = 6;       //local variables
  i = 10;
  System.out.println("i = " + i);
  i = method1(x, y);
  System.out.println("i = " + i);
  i = method2(x, y);
  System.out.println("i = " + i);
 }      //end main
```

```
12.
13.    static int method1(int arg11 ,int arg12 ){
14.     double num11 ,num12;
15.     for(int counter = 0; counter <= 5; counter++){
16.      i+= counter;
17.      }    //end of for counter loop
18.     return i+arg11+arg12;
19.    }      //end method1
20.
21.    static int  method2 (int  arg21, int arg22){
22.     int num21, num22, i=0;   //local variables
23.     {
24.      String s; //local variable
25.     }
26.     return i+arg21+arg22;
27.    }      //end method1
28.   }               //end of class VariableScope
```

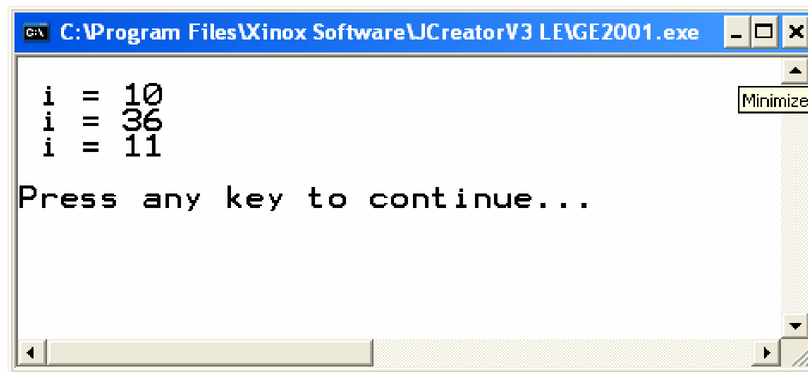
شرح المثال:

من خلال هذا المثال سوف نتعرف على أن مجال المتغير (Variable Scope) يؤثر على المكان الممكن استخدام هذا المتغير فيه. في السطر (٢) تم تعريف المتغير i ليكون مرئي على مستوى الصنف VariableScope كاملاً، حيث تكون فترة حياة هذا المتغير من بداية تحميل الصنف إلى نهايته، وبما أن هذا الصنف يحتوي على الطريقة ()main فإنه يعتبر الصنف الرئيسي لتنفيذ البرنامج، وبذلك تكون فترة حياة المتغير i من بداية البرنامج إلى نهايته). في السطر (٤) تم تعريف المتغيرين x و y كمتغيرات محلية (Local Variables) يمكن رؤيتها داخل الطريقة ()main فقط، وفترة حياتهما تمتد من بداية الطريقة ()main إلى نهايتها. في السطر (١٣) المعاملان arg11 و arg12 الخاصين بالطريقة method1 فيكونان مرئيان فقط داخل هذه الطريقة، وفترة حياتهما تبدأ من لحظة استدعاء الطريقة ولغاية الانتهاء من هذه الطريقة والخروج منها. في السطر (١٤) المتغيران num11 و num12 هما متغيران محليان ويكونان مرئيان داخل الطريقة method1 فقط، وتبدأ فترة حياتهما باستدعاء الطريقة وتنتهي بالخروج منها. في السطر (١٥) تم تعريف المتغير counter ليكون مرئي داخل جملة الدوران for فقط، وتمتد فترة حياة هذا المتغير من لحظة الدخول إلى جملة الدوران وتستمر حتى نهاية المقطع (block) الخاص بهذه الجملة. وفي السطر

(٢٢) تم تعريف متغيرات محلية للطريقة method2 ومن هذه المتغيرات متغير اسمه i ،ونلاحظ أن اسم هذا المتغير يتطابق مع اسم المتغير المعرّف على مستوى الصنف في السطر (٢)، وهذا التعريف يلغي رؤية المتغير i المعرّف على مستوى الصنف داخل هذه الطريقة. وعند استخدام المتغير داخل الطريقة فهذا يعني الرجوع للمتغير المعرّف على مستوى الطريقة فقط. وفي السطر (٢٤) تم تعريف المتغير s على مستوى المقطع (Block) الذي يبدأ من السطر (٢٣) وينتهي بالسطر (٢٥)، وبهذا يكون مجال رؤية هذا المتغير داخل هذا المقطع فقط وفترة حياته تبدأ من بداية المقطع وتنتهي بنهاية المقطع.

والآن سوف نشرح تنفيذ بعض جمل البرنامج، في السطر (٥) يتم اسناد الرقم ١٠ إلى المتغير i المعرّف على مستوى الصنف.، وبعد ذلك وفي السطر (٦) يتم طباعة محتويات المتغير i. وفي السطر (٧) يتم استدعاء الطريقة Method1 و إرسال ٥ و ٦ إلى معاملاتها x و y وبالترتيب.، وبعد ذلك ترجع هذه الطريقة ٣٦ ليتم تخزين هذا الرقم في i. ومن ثم يتم طباعة محتويات i من خلال تنفيذ الجملة في السطر (٨). وفي السطر (٩) يتم استدعاء الطريقة method2 بإرسال ٥ و ٦ لمعاملتها، بحيث ترجع هذه الطريقة ١١ ليتم تخزينه في i، ومن خلال السطر (١٠) يتم طباعة محتويات i. لاحظ أن قيمة i داخل الطريقة method2 هي صفر.

والشكل (٢-١٠) يبين نتائج تنفيذ هذا البرنامج.



شكل (٢-١٠)

انواع تمرير البيانات:

هنالك طريقتان لعملية تمرير البيانات إلى الطرق:

١ -التمرير باستخدام القيمة (Pass-By-Value): وفي هذا النوع يتم إرسال نسخة من قيمة المتغير (المعامل الفعلي) إلى معامل الطريقة (المعامل الشكلي) المقابل له. أي أن عملية التعديل على المعامل الشكلي لا تؤثر على المتغير (المعامل الفعلي) الذي تم إرساله إلى الطريقة عند الاستدعاء. حيث أن

هذا النوع من تمرير البيانات يتم تطبيقه تلقائياً عندما يكون نوع المعاملات الفعلية من الأنواع البدائية (Primitive Data Types) مثل: int، double، float، . . . الخ.

٢ -التمرير باستخدام العنوان (Pass-By-Reference): وفي هذا النوع يتم إرسال عنوان المتغير (المعامل الفعلي) إلى المعامل الشكلي المقابل له في الطريقة (Method)، ليصبح المعامل الشكلي والمعامل الفعلي يؤشران إلى نفس العنوان في الذاكرة الرئيسية. وفي هذا النوع أي تغيير يتم على المعامل الشكلي يؤثر وفي نفس الوقت على المعامل الفعلي الذي تم إرساله للطريقة عند الاستدعاء.وهذا النوع من تمرير البيانات يتم تطبيقه بشكل تلقائي عندما تكون المعاملات الفعلية من أحد الكائنات (Objects) مثل: (المصفوفات) Arrays، String، . . . الخ.

مثال (٢-٦):

```
// Passing_Parameters.java

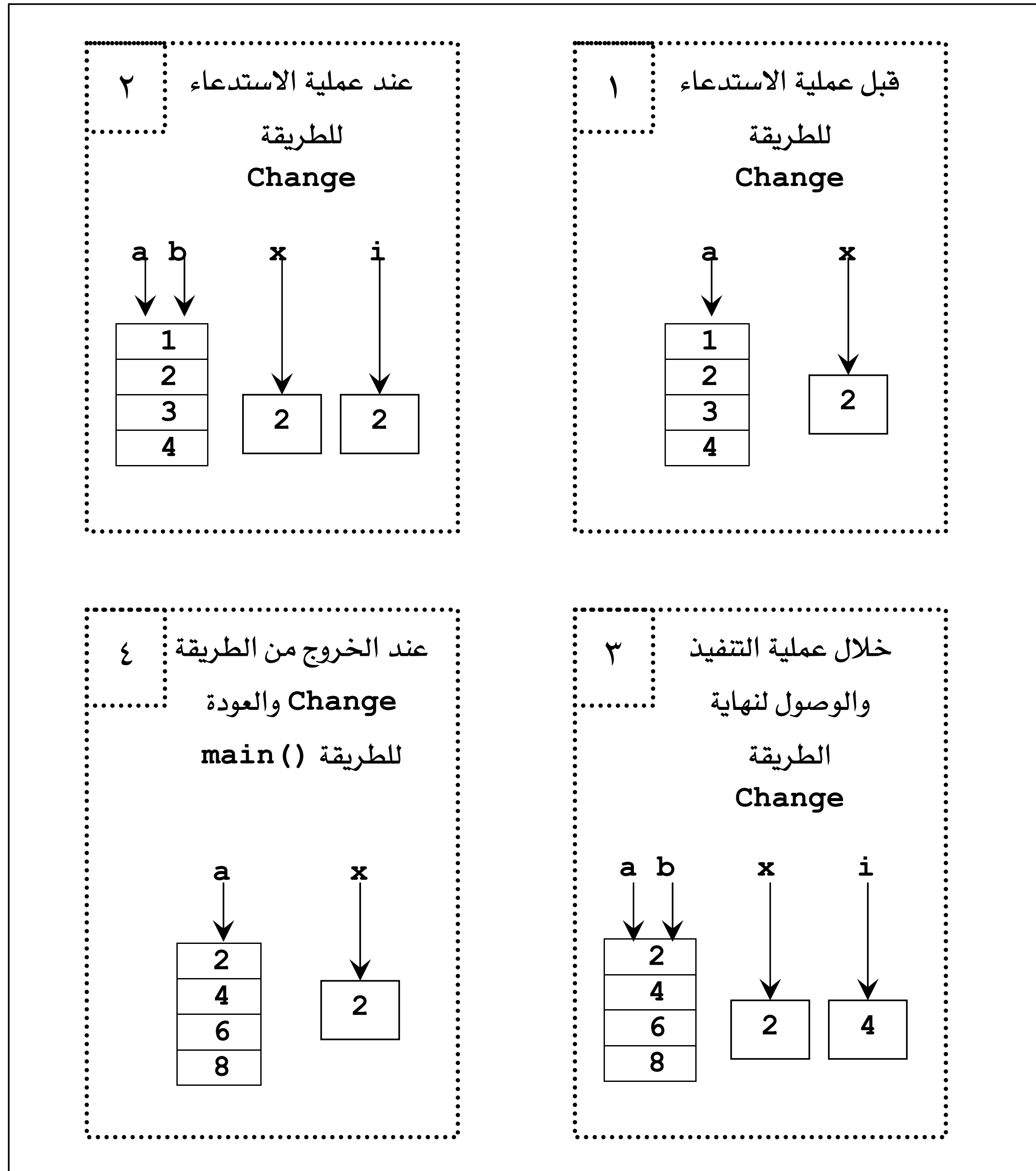
public class Passing_Parametres{
 public static void main(String args[]){
 int x;
 int a[] = {1, 2, 3, 4};
 x = a[1];
 System.out.println("The value of x before change is" + x) ;
 System.out.println("The value of a elements before change is: ");
 printArray(a);
 change(a, x);
     System.out.println("The value of x after change is" + x);
     System.out.println("The value of a elements after change is: ");
     printArray(a);


 } //end of main
  static void change(int b[], int i){
       i *= 2 ;
     for (int index=0; index < b.length; index++)
     b[index]*= 2;
 }        //end of method change
```

```
19.  static void printArray(int c[]){
20.     for (int index=0; index < c.length; index++)
21.     System.out.print(c[index] + "\t");
22.     System.out.println();
23.  }        //end of method print
24.  }        //end of class Passing_Parametres
```

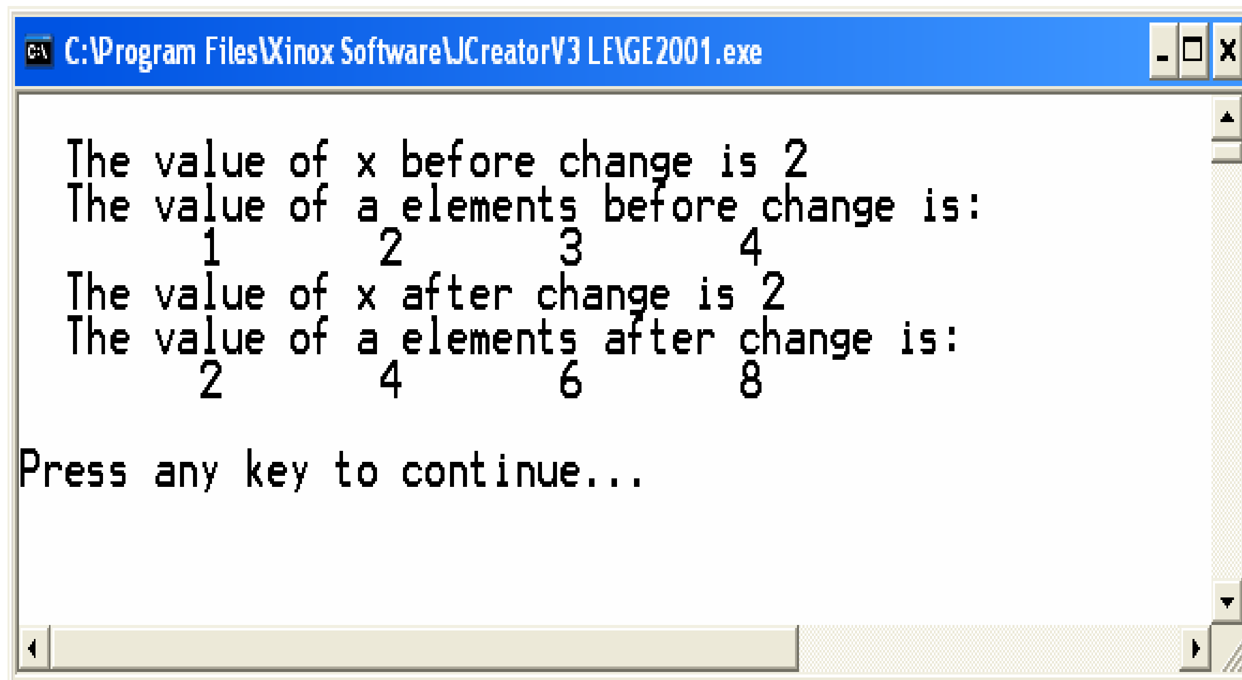
شرح المثال:

هذا المثال يبين كيفية الاستدعاء باستخدام القيمة والاستدعاء باستخدام العنوان. في السطر رقم (٩) تمت عملية استدعاء للطريقة change بإرسال x (متغير يشير إلى قيمة من نوع int) و a (متغير يشير إلى مصفوفة من نوع int). وكما ذكرنا سابقاً فإن عملية تمرير المتغيرات التي تشير إلى أنواع بيانات بدائية (Primitive Data Types) تكون باستخدام القيمة حيث سيتم إرسال نسخة من قيمة المعامل الفعلي x إلى المعامل الشكلي i الموجود في تعريف هذه الطريقة. وبما أنه قد تم إرسال نسخة من قيمة المعامل الفعلي إلى المعامل الشكلي فهذا يعني بأنهما يشيران إلى مكانين مختلفين في الذاكرة الرئيسية. أما بالنسبة للمعامل الشكلي الثاني a والذي يرسل عند استدعاء الطريقة change فهو عبارة عن مصفوفة. وكما نعرف بأن المصفوفة هي عبارة عن كائن (Objects) إذن سيتم إرسال عنوان هذه المصفوفة إلى المعامل الشكلي b، أي سيكون المعامل الفعلي a والمعامل الشكلي b يشيران إلى نفس الموقع (المصفوفة) في الذاكرة. والشكل (٢-١١) يوضّح الفرق بين عملية التمرير باستخدام القيمة وعملية التمرير باستخدام العنوان وذلك عند استدعاء الطريقة change الموجودة في السطر (٩) في المثال السابق.


١
قبل عملية الاستدعاء للطريقة Change
a
x
1
2
3
4
2
٢
عند عملية الاستدعاء للطريقة Change
a b
x
i
1
2
3
4
2
2
٣
خلال عملية التنفيذ والوصول لنهاية الطريقة Change
a b
x
i
2
4
6
8
2
4
٤
عند الخروج من الطريقة Change والعودة للطريقة main()
a
x
2
4
6
8
2

شكل (٢-١١)

والشكل (٢-١٢) يبين نتائج تنفيذ البرنامج في المثال (٢-٦).



شكل (٢-١٢)

الاستدعاء الذاتي (Recursion):

والمقصود بالاستدعاء الذاتي هو أن تقوم الطريقة باستدعاء نفسها بنفسها ، فهنالك الكثير من المسائل التي يمكن حلها باستخدام الاستدعاء الذاتي وبهذه الحالة يمكن توفير الكثير من جمل التكرار فنستطيع الاستعاضة عن جمل التكرار بعملية استدعاء الطريقة لنفسها. فمثلا لإيجاد المضروب (factorial) لعدد معين يمكن كتابة البرنامج على الشكل التالي :

مثال (٢-٧):

```
// factorial.java

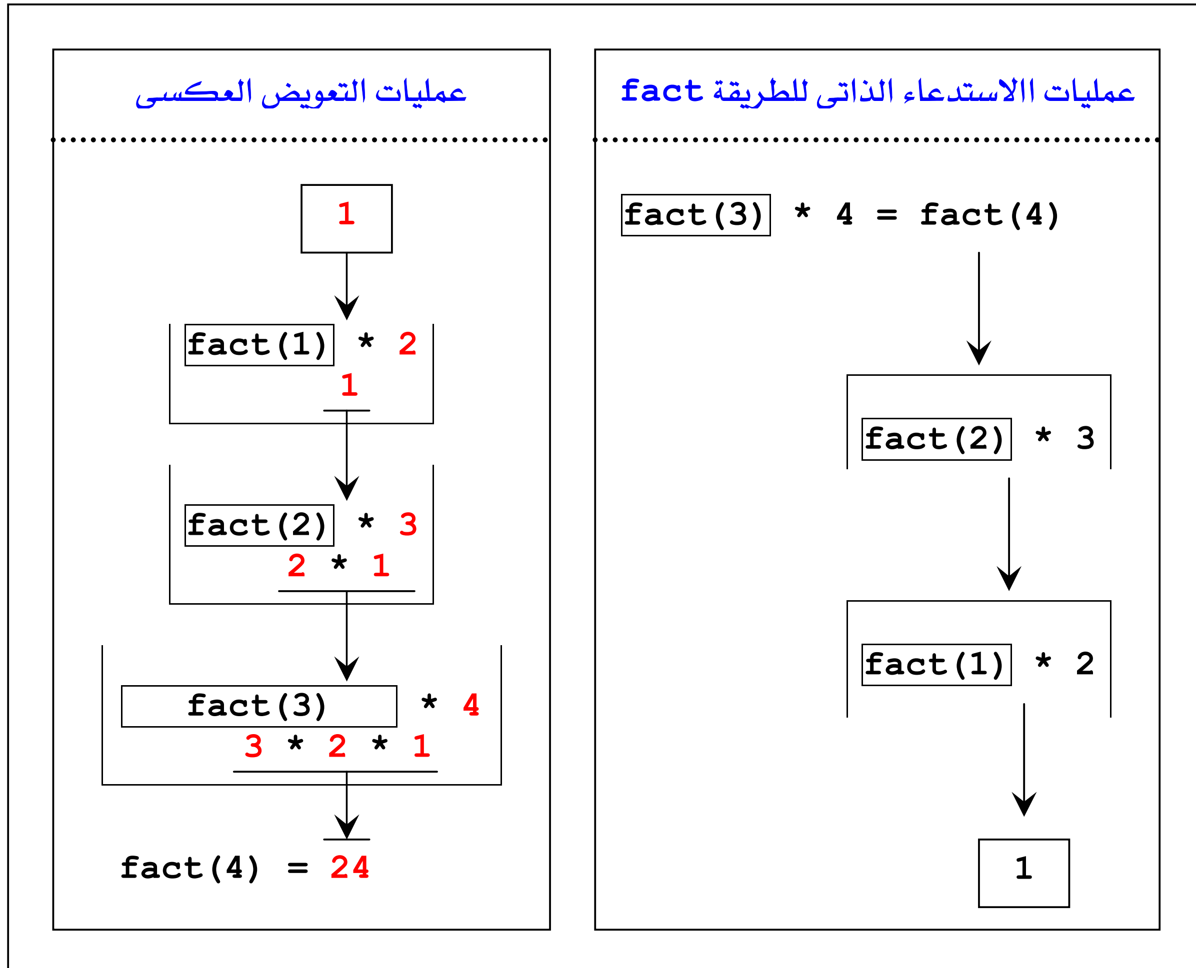
import javax.swing.JOptionPane;
public class factorial {
 public static void main (String args[ ]){
  String snum1;
```

```
5.      int num1, fac_of_num1;
6.      snum1 = JOptionPane.showInputDialog("Enter num1:") ;

7.     num1 = Integer.parseInt(snum1);
8.      fac_of_num1 = fact(num1);
9.     JOptionPane.showMessageDialog(null, num1 + "! = " +
                                               fac_of_num1);
10.    }      //end of main
11.
12.    static int fact(int n){
13.     if (n == 0 || n ==1)
14.       return 1;
15.     else
16.       return n * fact(n-1);
17.   }             //end of fact method
18.   }             //end of class factorial
```

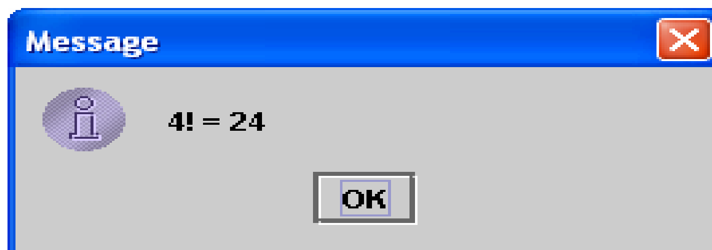
شرح المثال:

في السطر (٦) يتم إدخال الرقم المراد حساب قيمة المضروب (Factorial) له. ثم في السطر (٨) يتم استدعاء الطريقة fact والتي تقوم بعملية حساب قيمة المضروب وتعيدها ليتم تخزينها في المتغير fac_of_num1. والآن لنرى ماذا سيحدث عند استدعاء الطريقة fact: في الاسطر (١٣-١٤) تتم عملية السؤال عن قيمة الرقم المرسل فإذا كان مساوياً للواحد أو للصفر (حسب التعريف الرياضي لعملية إيجاد المضروب) تتوقف هذه الطريقة وترجع واحد. ويعتبر هذا الشرط هو شرط التوقف الوحيد للاستدعاء الذاتي لهذه الطريقة، حيث أنه لابد من وجود شرط توقف داخل طرق الاستدعاء الذاتي وإلا استمرت عملية الاستدعاء الذاتي إلى مالانهاية. وفي السطر (١٦) تتم ارجاع العدد مضروباً بعملية استدعاء أخرى لنفس الطريقة ولكن هذه المرة بإرسال العدد السابق مطروحاً منه واحد (n-1)، وتستمر هذه العملية حتى يتحقق شرط الخروج من الطريقة وتوقيف عملية الاستدعاء الذاتي وتبدأ الآن عملية التعويض العكسي للقيم المرجعة من عمليات الاستدعاء. والشكل (٢-١٣) يوضّح عمليات الاستدعاء وعمليات التعويض العكسي عند إدخال الرقم ٤.



شكل (٢-١٣)

والشكل (٢-١٤) التالي يبين مخرجات هذا البرنامج:



شكل (٢-١٤)

التحميل الزائد للطرق (Methods Overloading):

تتم عملية التحميل الزائد للطريقة عندما يكون هنالك أكثر من طريقة تحمل نفس الاسم في نفس الصنف ويتم التمييز بين هذه الطرق من خلال التوقيع (Signature) الخاص بكل منهم، أي إذا أردنا أن نقوم بتعريف أكثر من طريقة بنفس الاسم لابد أن تختلف هذه الطرق في: عدد المعاملات، نوع المعاملات، أو ترتيب المعاملات المختلفة الأنواع.

مثال (٢-٨):

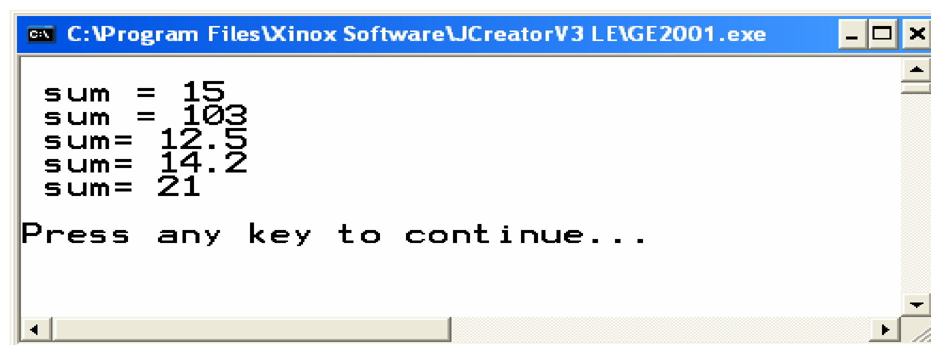
```
// Overload.java

public class Overload {
public static void main(String args[]){
sum();
sum(100,3);
System.out.println("sum= " + sum(8.5, 4));
System.out.println("sum= " + sum(10, 4.2));
System.out.println("sum= " + sum(8, 9, 4));
}
// no parametrs no return
static  void sum () {
 int num1 = 10, num2 = 5;
 System.out.println("sum = " + (num1 + num2));
}
// has two parametes and no return
static  void sum (int num1, int num2) {
 System.out.println("sum = " + (num1 + num2));
}
// has two parametes   and return double
static double sum( double num1, int num2) {
 return (double)(num1 + num2);
}
// has two parameters and return double
// but different order of the parameters
static double sum(int num2 ,double num1) {
```

```
27.    return (double)(num1 + num2);
28.   }
29.
30.   // has three parameters and return integer
31.   static int sum(int num1, int num2, int num3) {
32.    return num1+num2+num3;
33.   }
34.   }
```

شرح المثال:

يوضّح هذا المثال مفهوم التحميل الزائد للطرق (Methods Overloading). نلاحظ في هذا المثال وجود خمس طرق تحمل نفس الاسم sum، لكن لابد لهذه الطرق أن تختلف عن بعضها في واحد أو أكثر من مكونات التوقيع (وتحديداً يكون الاختلاف بنوع المعاملات، عدد المعاملات، أو ترتيب المعاملات المختلفة النوع). ففي السطر (١١) تم تعريف الطريقة sum حيث أنه ليس لها معاملات شكلية ولا ترجع قيمة من اي نوع (void). وفي السطر (١٦) تم تعريف طريقة ثانية لها نفس الاسم sum بحيث تحتوي على معاملين شكليين اثنين من نوع int ولا ترجع قيمة من أي نوع (void). بينما في السطر (٢١) تم تعريف طريقة ثالثة بنفس الاسم sum لكنها تحتوي على معاملين شكليين اثنين الاول من نوع double والثاني من نوع int ترجع هذه الطريقة قيمة من نوع double. وفي السطر (٢٦) تم تعريف طريقة رابعة لها نفس الاسم sum ولها معاملين شكليين الأول من نوع int والثاني من نوع double وترجع قيمة من نوع double. واخيراً في السطر (٣١) تم تعريف طريقة خامسة لها الاسم sum وتحتوي على ثلاث معاملات شكلية من نوع int وترجع قيمة من نوع int. والشكل (٢-١٥) يبين مخرجات هذا البرنامج بعد تنفيذه.



شكل (٢-١٥)

الطرق الخاصة بالسلاسل الرمزية (String):

السلاسل الرمزية(String) هي عبارة عن مجموعة من الرموز characters (حروف، ارقام، أو رموز خاصة) المتجاورة، وفي معظم لغات البرمجة تكون السلاسل الرمزية (String) على شكل مصفوفة من الرموز characters، ولكن في java تكون السلاسل الرمزية ككائن منفصل من الصنف المسمى String ويحتوي هذا الصنف على العديد من الطرق الخاصة بمعالجة هذا النوع من البيانات ولكن هناك عملية واحدة يمكن استعمالها على السلاسل الرمزية (String) دون الحاجة إلى استدعاء أي طريقة ألا وهي عملية الدمج "+"، حيث تقوم هذه العملية بدمج سلسلتين رمزيتين لتكوين سلسلة رمزية واحدة. وتتم عملية تعريف المغيرات من نوع السلاسل الرمزية (String) وذلك باستخدام اسم الصنف String بدل نوع البيانات عند تعريف متغيرات من نوع بدائي (Primitive Data Types) كما في المثال التالي:

```
1. String s;
2. s= "Hello";
```

حيث تم في السطر (١) تعريف المتغير s من نوع الصنف String، وهنا يعتيبر s ككائن من نوع الصنف String. وفي السطر (٢) تم تخزين قيمة ثابتة هي "Hello" من نوع السلاسل الرمزية في المتغير s. والجدول (٢-٢) يحتوي شرح لبعض الطرق (Methods) الخاصة بالسلاسل الرمزية.

إيجاد طول السلسلة الرمزية:	
`s.length()`	ترجع الطريقة `length()` طول السلسلة الرمزية **s**.
عمليات المقارنة بين سلستين رمزيتين (ملاحظة: لاتستخدم `==` و `!=`).	
`s.compareTo(t)`	تقوم الطريقة بمقارنة السلسلة الرمزية **s** مع السلسلة الرمزية **t** وتعيد رقم سالب اذا كانت **s** اقل من **t** وتعيد صفر إذا كانت **s** تساوي **t** وتعيد رقم موجب إذا كانت **s** أكبر من **t**.
`s.compareToIgnoreCase(t)`	تعمل هذه الطريقة بنفس عمل الطريقة

	compareTo() ولكن مع اهمال حالة الحروف (صغيرة أم كبيرة).
s.equals(t)	تعيد true إذا كان s يساوي t.
s.equalsIgnoreCase(t)	تعمل هذه الطريقة بنفس عمل الطريقة equals() ولكن مع إهمال حالة الحروف (صغيرة أم كبيرة).
s.startsWith(t)	تعيد true إذا كان s يبدأ بالسلسلة الرمزية t.
s.startsWith(t, i)	تعيد true إذا كانت السلسلة الرمزية t موجودة في s بدءاً من الموقع i.
s.endsWith(t)	تعيد true إذا كان s تنتهي بـ t.

عمليات البحث:

كل طرق indexOf() تقوم بإرجاع 1- إذا كان العنصر المراد البحث عنه غير موجود، ويمكن للعنصر المراد البحث عنه أن يكون حرف أو سلسلة رمزية.

s.indexOf(t)	ترجع موقع أول مكان توجد فيه t داخل السلسلة الرمزية s.
s.indexOf(t, i)	ترجع موقع أول مكان توجد فيه t داخل السلسلة الرمزية s بعد الموقع i.
s.indexOf(c)	ترجع موقع أول مكان يوجد فيه الحرف المخزن في المتغير c داخل السلسلة الرمزية s.
s.indexOf(c, i)	ترجع موقع أول مكان يوجد فيه الحرف المخزن في المتغير c داخل السلسلة الرمزية s بعد الموقع i.
s.lastIndexOf(c)	ترجع موقع آخر مكان يوجد فيه الحرف المخزن في المتغير c داخل السلسلة الرمزية s.

`s.lastIndexOf(t)`	ترجع موقع آخر مكان توجد فيه السلسلة الرمزية **t** داخل السلسلة الرمزية **s**.
عمليات أخذ جزء من السلسلة الرمزية **string**.	
`s.charAt(i)`	ترجع الحرف الموجود في الموقع **i** داخل السلسلة الرمزية **s**.
`s.substring(i)`	ترجع جزء من السلسلة الرمزية **s** بدءاً من الموقع **i** وحتى النهاية.
`s.substring(i, j)`	ترجع جزء من السلسلة الرمزية **s** بدءاً من الموقع **i** وحتى الموقع **j-1**.
عمليات التعديل على السلسلة الرمزية **string** وإنشاء سلسلة رمزية جديدة.	
`s.toLowerCase()`	إنشاء سلسلة رمزية جديدة تحتوي كل ما في السلسلة الرمزية **s** بعد تحويل كل الحروف إلى حروف صغيرة.
`s.toUpperCase()`	إنشاء سلسلة رمزية جديدة تحتوي كل ما في السلسلة الرمزية **s** بعد تحويل كل الحروف إلى حروف كبيرة.
`s.trim()`	إنشاء سلسلة رمزية جديدة من السلسلة الرمزية **s** بعد الفارغ من البداية والنهاية.
`s.replace(c1, c2)`	إنشاء سلسلة رمزية جديدة من السلسلة الرمزية **s** بعد تبديل كل **c1** بـِ **c2**، وهما من نوع **char**.
عمليات أخرى على السلاسل الرمزية **string**.	
`s.matches(regexStr)`	ترجع هذه الطريقة **true** إذا كانت السلسلة

	الرمزية **regexStr** تطابق السلسلة الرمزية **s** كاملة.
`s.replaceAll(regexStr, t)`	إنشاء سلسلة رمزية **string** جديدة بعد تبديل كل **regexStr** بـ **t**.
`s.replaceFirst(regexStr, t)`	إنشاء سلسلة رمزية **string** جديدة بعد تبديل أول **regexStr** بـ **t**.
`s.split(regexStr)`	إنشاء مصفوفة تحتوي على أجزاء من السلسلة الرمزية **s** مقسمة حسب ظهور **regexStr**.
`s.split(regexStr, count)`	كما في الطريقة **split(regexStr)** لكن مع تحديد عدد مرات التقسيم.

جدول (٢-٢)

والمثال (٢-٩) التالي يوضّح تنفيذ معظم الطرق الموجودة في الجدول (٢-٢) السابق.

مثال (٢-٩):

```
// Strings.java
public  class UseMath{
 public static void main(String args[]){
  String s0="Well Come to Java World!" ,
  s = "hello", t = "HELLO", s1, s2[], s3;
  char c;
  boolean b;
  int i;
  System.out.println();
  i = s0.length();
   System.out.println(" The length of " + "\"" + s0 + "\"" + " = " + I +
                                          "\n");
   i = s.compareTo(t);
   if (i == 0)
   System.out.println(" \"" + s + "\"" + " is == " + "\"" + t + "\"\n");
```

```
else if (i<0)
System.out.println(" \"" + s + "\"" + " is < " + "\"" + t + "\"\n");
else
System.out.println(" \"" + s + "\"" + " is > " + "\"" + t +"\"\n");
i = s.compareToIgnoreCase(t);
System.out.print(" Ignoring case: ");
if (i == 0)
System.out.println(" \"" + s + "\"" + " is == " + "\"" + t + "\"\n");
else if (i<0)
System.out.println(" \"" + s + "\"" + " is < " + "\"" + t + "\"\n");
else
System.out.println(" \"" + s + "\"" + " is > " + "\"" + t +"\"\n");
b =  s.equals(t);
System.out.println(" Is " + "\"" + s + "\"" + " equals to " + "\"" + t
        + "\"" + " ?  " + b +"\n");
b = s.equalsIgnoreCase(t);
System.out.print(" Is " + "\"" + s + "\"" + " equals to ");
System.out.println("\"" + t + "\"" + " (ignoring case)? " + b +
        "\n");
b = s.startsWith("H");
System.out.println(" Is " + "\"" + s + "\"" + " starts with \"H\"? " +
        b + "\n");
b = s.startsWith("l", 3);
System.out.print(" Is " + "\"" + s + "\"" + " starts with \"l\" ");
System.out.println("from position 3 ? " + b + "\n");
b = s.endsWith("lo");
System.out.print (" Is " + "\"" + s + "\"" + " ends with \"lo\"");
System.out.println(" ,from position 3 ? " + b + "\n");
i = s0.indexOf("Java");
System.out.print (" Java is at position ");
System.out.println( i + " of " + "\"" + s0 + "\"\n");
i = s0.indexOf("java", 4);
System.out.print(" java is at position" + i +"of ");
System.out.println(" \"" + s0 + "\"" + ", starting from position
        4\n");
i = s0.indexOf('e');
System.out.print (" \'e\' is at position ");
```

```
System.out.println(i + " of " + "\"" + s0 + "\"\n");
i = s0.indexOf('e', 4);
System.out.print (" \'e\' is at position " + i + " of  ");
System.out.println("\"" + s0 + "\"" + " starting from position 4\n");
i = s0.lastIndexOf('e');
System.out.print(" Last occurrence  of \'e\' in ");
System.out.println("\"" + s0 + "\"" + " is at "+ I + "\n");
i = s0.lastIndexOf(t);
System.out.print(" Last occurrence  of 'rl' in ");
System.out.println("\"" + s0 + "\"" + " is at " + I + "\n");
c = s0.charAt(3);
System.out.print(" The character at position 3 in");
System.out.println("\"" + s0 + "\"" + " is  " + c + "\n");
s3 = s0.substring(6);
System.out.print (" The substring of ");
System.out.println("\"" + s0 + "\"" + " starting from 6 is\n" +
                                    "\t\t\t\"" + s3 + "\"\n");
s1 = s0.substring(6, 10);
System.out.print(" Substring of " + "\"" + s0 + "\"" + " starting ");
System.out.println("from 6 to 10 is:" + "\"" + s1 + "\"\n");
System.out.print (" \"" + s0 + "\"" + " in lowercase is ");
System.out.println("\"" + s0.toLowerCase() + "\"\n");
System.out.print("\""+ s0 + "\"" + "in uppercase ");
System.out.println("\""+ s0.toUpperCase() + "\"\n");
System.out.print(" \"" + s0 + "\"" + " with replacing all spaces ");
System.out.println("with ';' is\n" + "\t\t\t\"" + s0.replace( '
                                    ',';')+"\"");
System.out.println();
}
}
```

والشكل (٢-١٦) يبين مخرجات هذا البرنامج الذي يوضّح بعض الطرق الخاصة بالسلاسل الرمزية والموجودة في الصنف (String).

```
The length of "Well Come to Java World!" = 24
"hello" is > "HELLO"
Ignoring case:  "hello" is == "HELLO"
Is "hello" equals to "HELLO" ?  false
Is "hello" equals to "HELLO" (ignoring case)? true
Is "hello" starts with "H"? false
Is "hello" starts with "l" from position 3 ? true
Is "hello" ends with "lo" ,from position 3 ? true
Java is at position 13 of "Well Come to Java World!"
java is at position -1 of  "Well Come to Java World!", starting from position 4
'e' is at position 1 of "Well Come to Java World!"
'e' is at position 8 of  "Well Come to Java World!" starting from position 4
Last occurrence  of 'e' in "Well Come to Java World!" is at 8
Last occurrence  of 'rl' in "Well Come to Java World!" is at -1
The character at position 3 in "Well Come to Java World!" is  l
The substring of "Well Come to Java World!" starting from 6 is
                        "ome to Java World!"
Substring of "Well Come to Java World!" starting from 6 to 10 is:"ome "
"Well Come to Java World!" in lowercase is "well come to java world!"
"Well Come to Java World!" in uppercase "WELL COME TO JAVA WORLD!"
"Well Come to Java World!" with replacing all spaces with ';' is
                        "Well;Come;to;Java;World!"
Press any key to continue...
```

شكل (٢-١٦)

تمارين:

س١: اكتب برنامجاً تطبيقياً بلغة جافا لإيجاد مساحة الدائرة:

$$Area = r^2 X \pi$$

(r : نصف القطر)

س٢: وضح الفرق بين المعاملات الشكلية (Formal parameters) والمعاملات الفعلية (Actual parameters or arguments).

س٣: وضح المقصود بما يلي:

أ - public method

ب - private method

س٤: ما هي الآلية التي من خلالها يتم إرجاع البيانات من الطريقة.

س٥: وضح باستخدام الأمثلة ما المقصود بما يلي:

أ - التمرير باستخدام القيمة Pass-By-Value

ب - التمرير باستخدام العنوان Pass-By-Reference

س٦: من المعلوم أن مجموع الأعداد من واحد إلى N يمثل بالمعادلة التالية:

$$\sum_{i=1}^{N} i = N + \sum_{i=1}^{N-1} i = N + N - 1 + \sum_{i=1}^{N-2} i$$

اكتب برنامجاً تطبيقياً بلغة جافا لحل هذه المعادلة باستخدام:

أ - الاستدعاء الذاتي.

ب - جمل التكرار (الدوران).

س٧: اكتب برنامجاً تطبيقياً بلغة جافا لإيجاد حاصل ضرب iXj (ضرب الأعداد الصحيحة) حيث إن i>0، وذلك باستخدام عملية الجمع، مثلاً: 3X4 = 3+3+3+3 = 12.

س٩: اكتب طريقة تستقبل مصفوفة أعداد صحيحة وتعيد true إذا كانت جميع عناصر المصفوفة أعداد زوجية وتعيد false إذا كانت غير ذلك.

س١٠: وضح باستخدام مثال المقصود بالمصطلح Method Overloading.

س١١: اكتب برنامجا تطبيقيا بلغة جافا يحتوي على طريقتين (methods) الأولى تقوم بعملية تحويل درجات الحرارة المئوية Celsius إلى فهرنهايت Fahrenheit، حيث إن معادلة التحويل من المئوي إلى الفهرنهايتي هي:

F = 9.0 / 5.0 * (C + 32)

والطريقة الاخرى تقوم بعملية التحويل من الفهرنهايت Fahrenheit إلى المئوي Celsius، والمعادلة التالية تستخدم للتحويل من الفهرنهايتي إلى المئوي:

C = 5.0 / 9.0 * (F – 32)

س١٢: اكتب برنامجاً تطبيقياً بلغة جافا يحتوي على الطرق التالية (جميع الطرق تستقبل متغير من نوع (String):

أ -طريقة تعيد عدد الجمل في السلسلة الرمزية (يتم الفصل بين الجمل عند الادخال بالنقطة).

ب -طريقة تعيد عدد الكلمات في السلسلة الرمزية (يتم الفصل بين الكلمات بالفراغ).

ج -طريقة تعيد عدد الكلمات في كل جملة من الجمل التي تم معرفتها من خلال الطريقة في الفقرة (أ).

د -طريقة تعيد عدد الأحرف في كل جملة من الجمل التي تم معرفتها من خلال الطريقة في الفقرة (أ)، دون احتساب الفراغ أو علامات الترقيم.

هـ -طريقة تعيد متوسط عدد الكلمات لكل الجمل.

و -طريقة تعيد متوسط عدد الأحرف لكل الكلمات.

س١٣: اكتب ناتج تنفيذ البرامج التالية:

أ-

```
public class checkupper {
 public static void main (String args[]){
  char c1 = 'f', c2 = 'T';
  System.out.println("Is "+c1 +" in uppercase ? " +
                          isUpperCase(c1));
  System.out.println("Is "+c2 +" in uppercase ? " +
                          isUpperCase(c2));
 }
 static boolean isUpperCase(char testChar) {
  return ((testChar>='A') && (testChar<='Z'));
  }
}
```

ب –

```
public class validateAddress {
 public static void main (String args[]){
  String My_email = "java_doc@java.net";
  if (validate(My_email) == true)
   System.out.println("this a valid email address");
  else
   System.out.println("this an invalid email
                          address");
 }
 static boolean validate(String email) {
  String name;
  String domain;
```

```
  int index;
  if (( index = email.indexOf( '@')) == -1) {
   return false;
  }
  name = email. substring(0, index);
  domain=email.substring(index+1,email.length());
  System.out.println(" Name: " + name);
  System.out.println(" Domain: " + domain);
  return true;
 }
}
```

ج-

```
public class primenumbers{
 public static void main(String [] args) {
  System.out.println("The Prime numbers between 1 and
                     100 are");
  for (int i = 0; i < 100; i++)
   if (isPrime(i))
    System. out. print(i + " ");
 }
 static boolean isPrime(int test) {
  if (test < 2)
   return false;
  if (test == 2)
     return true;
  for (int i = 2; i < test; i++)
```

```
      if (( test % i) == 0)
        return false;
  return true;
 }
}
```

د-

```
public class SwapArray{
 public static void main(String [] args) {
  int values[]={1, 2, 3, 4, 5, 6, 7, 8};
  System.out.println("values before swap");
  printArray(values);
  swap(values);
  System.out.println ("values after swap");
  printArray(values);
 }
 static void swap(int a[]){
  int length = a.length, temp;
  for (int i = 0; i <= (length/2); i++){
   temp = a[ length - i - 1];
   a[length - i - 1] = a[ i];
   a[ i] = temp;
  }
 }
 static void printArray(int a[]){
  for (int i =0;i<a.length;i++){
   System.out.print (a[i]+"\t");
  }
```

```
  System.out.println( );
 }
}
```



برمجة ٢

الأصناف والكائنات

الأصناف والكائنات

٣

الجـــــدارة:

أن يكون المتدرب قادراً على التعامل مع الأصناف والكائنات في كتابة برامج لغة جافا.

الأهــــداف:

عندما تكمل هذه الوحدة تكون قادراً على:

١ - فهم ماهية البرمجة الكينونية (OOP) وكيفية الاستفادة منها.

٢ - تعريف الاصناف وتحديد مكوناتها من بيانات وطرق.

٣ - فهم واستخدام مفهوم الوراثة للأصناف وكيفية الوصول للطرق الموروثة.

٤ - استبدال الطرق الموروثة بطرق أخرى (Method Overriding).

مستوى الأداء المطلوب:

أن يصل المتدرب إلى إتقان هذه الجدارة بنسبة ١٠٠٪.

الوقت المتوقع للتدريب: ٨ ساعات.

الوسائل المساعـدة:

- قلم.
- دفتر.
- جهاز حاسب آلي.

متطلبات الجدارة:

اجتياز جميع الحقائب السابقة.

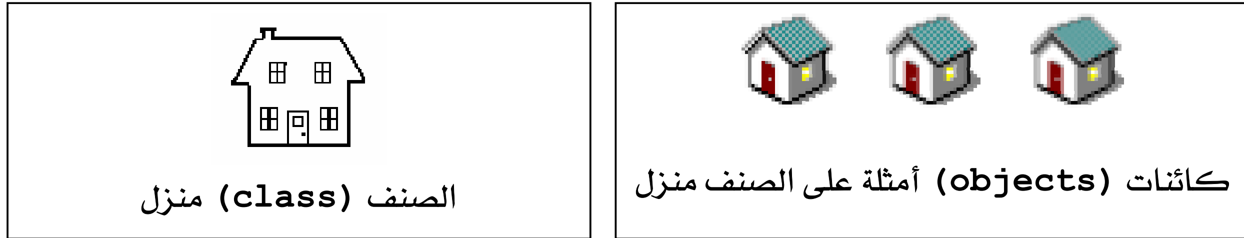
البرمجة الموجهة للكائنات (Object Oriented Programming):

تعتبر الأصناف (classes) والكائنات (objects) مفهومين أساسين في برمجة الكائنات وتكمن الفائدة في استخدام برمجة الكائنات في أن معظم برامج الحاسوب المستخدمة لحل المشاكل الحقيقية تكون كبيرة جدا وأكبر من تلك التي المستخدمة في الوحدات السابقة وثبت عمليا أن أفضل طريقة لكتابة البرنامج هي تقسيمه إلى وحدات صغيرة (modules) ويعرف هذا المبدأ بمبدأ "فرّق تسد" (divide and conquer).

وحدات البرنامج في لغة جافا هي الأصناف (classes). عندما يقوم الشخص بكتابة برنامج جديد يقوم بضم الأصناف (classes) الجديدة مع تلك المتوفرة في مكتبة الأصناف في جافا (API) ،وتتم عملية التخاطب بين هذه الأصناف عن طريق الرسائل (messages). حيث توفر هذه المكتبة العديد من الأصناف التي تقوم بالعمليات الحسابية ومعالجة النصوص وعمليات الإدخال والإخراج والكثير الكثير من العمليات المفيدة الأخرى.

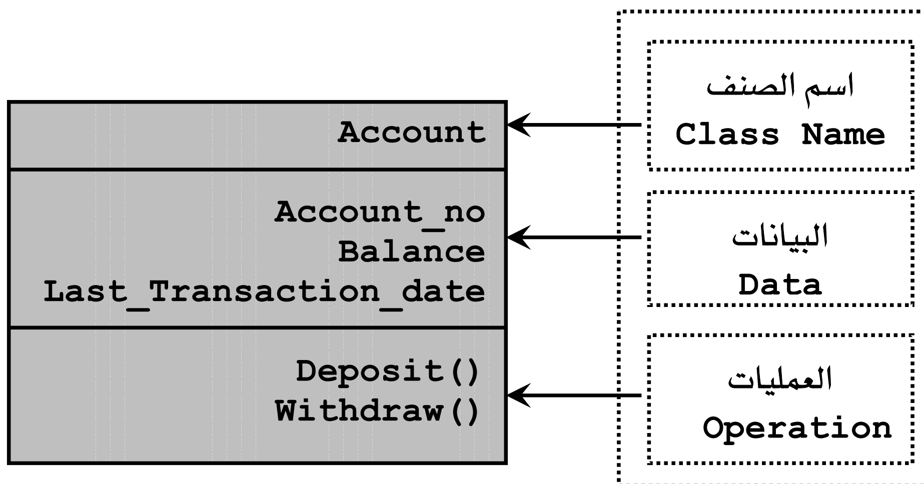
الكائن: هو عبارة عن شيء حقيقي في واقع الحياة العملية مثل الطالب محمد محمود احمد، الحساب رقم ١٢٢٣٢، السيارة التي تحمل اللوحة ك ا ن ١٠١، المريض خالد حسين . . . الخ، وكل هذه الأشياء تعتبر كائنات في بيئات مختلفة، فمثلا الطالب في البيئة الجامعية، الحساب في نظام البنك، السيارة في إدارة المرور، المريض في مستشفى، والى غير ذلك من الكائنات في بيئات العمل المختلفة.

الصنف: عبارة عن قالب (مخطط) يحتوي ويمثل الصفات للكائنات التي تنتمي لهذا الصنف، ويجب أن يحتوي هذا المخطط على جميع صفات الكائنات التي تنتمي إلية وجميع التفاصيل الخاصة بإنشاء هذه الكائنات (النسخ). .فمثلا الصنف "طالب" (Student) يمثل جميع الصفات لجميع الطلاب في بيئة معينة. والصنف "حساب" (Account) يمثل صفات جميع الحسابات في بنك معين. وهذه الصفات أو البيانات (Data) يتم تمثيلها في الأصناف بالمتغيرات بينما العمليات (Operation) يتم تمثييلها باستخدام الطرق (Methods). والشكل (٣-١) يبين العلاقة بين الصنف "منزل" وبين الكائنات الممكن أن تتبع له.



شكل (٣-١)

والشكل (٣-٢) يبين كيفية تمثيل الأصناف بشكل رسومي من خلال ما يسمى بالـ Class Diagram، حيث يبين هذا الرسم شكل بسيط جداً من الصنف "الحساب البنكي"، والذي يحتوي على البيانات والعمليات.



شكل (٣-٢)

تعـــريف الصنـــف (Class Declaration) وتحديـــد مكونـــاته:

يتم تعريف الأصناف في لغة جافا عن طريق استخدام الكلمة المحجوزة class، حيث يتبعها اسم الصنف، وعند اختيار اسم للصنف لابد من من تطبيق القواعد الخاصة بالأسماء (مثل: أسماء المتغيرات و أسماء الطرق) في لغة جافا. والمثال (٣-١) يبين كيفية تعريف الصنف Account، لكن دون وجود جمل تنفيذية لأنه للتوضيح فقط.

مثال (٣-١):

```
// Account.java
1.   import java.util.Date;
2.   public class Account {
3.     private int account_number;
4.     private double balance ;
5.     private Date last_transaction_date;
6.     public void deposit(double amt){
7.       //deposit code
8.     }
9.     public void withdraw(double amt){
10.      // withdraw code
11.    }
12.  }
```

تعريف الصنف

تعريف المتغيرات data

تعريف العمليات Operations

شرح المثال:

كما نلاحظ في هذا المثال فإن عملية تعريف الأصناف تكون بالطريقة التالية:

-نبدأ باسم الصنف (class name) ويمكن أن يكون مسبوقا بكلمة public (وتعني عام) وهذا يعني أنه يمكن لأي صنف آخر أن يقوم بإنشاء نسخ (instances) من هذا الصنف، أما إذا لم نضع كلمة public في عملية التعريف فإن الأصناف داخل نفس الحزمة (Package) التي يوجد بها هذا الصنف هي وحدها تستطيع إنشاء نسخ (instances) من هذا الصنف .

-ثم بعد ذلك نبدأ بتعريف المتغيرات كما في الاسطر (٣-٥)، و كما نلاحظ فإن هذه المتغيرات مسبوقة بكلمة private (وتعني خاص) وهذا يعني أن هذه المتغيرات يمكن التعامل معا داخل هذا الصنف فقط، أما إذا كانت مسبوقة بكلمة Public فإن جميع الأصناف يمكنها التعامل مع هذه المتغيرات (بعد إنشاء نسخة instance من هذا الصنف) أما إذا لم نضع شيء فإن الأصناف داخل نفس الحزمة (Package) التي يوجد بها هذا الصنف هي وحدها تستطيع التعامل مع هذه المتغيرات (بعد إنشاء نسخة instance من هذا الصنف).

-وفي الأسطر (٧-٩) و الأسطر (١٠-١٢) تم تعريف العمليات (الطرق) على الصنف.

إنشاء الكائن (Object Creation) والوصول لمكوناته :

والآن بعد أن لاحظنا كيف يتم تعرف الأصناف لنرى كيف يتم استخدام هذه الأصناف: تتم عملية استخدام الأصناف وذلك عن طريق إنشاء كائنات (Objects) تكون على شكل نسخ من هذا الصنف) وبالتالي يتم التعامل مع هذه الكائنات (النسخ)، .وتتم عملية انشاء النسخ على النحو التالي:

-تعريف متغير من نوع الصنف المراد استخدامه والذي تم تعريفه مسبقاً.

-إنشاء كائن حقيقي من نفس الصنف وذلك باستخدام كلمة new متبوعة بإحدى البانيات (Constructors).

-ثم بعد ذلك يتم التعامل مع الكائن باستخدام اسم المتغير الذي يشير إليه متبوعا بنقطة ثم بأحد المتغيرات أو الطرق حسب امكانية الوصول (public, private, protected, default). حيث تم شرح طرق الوصول هذه في الوحدة الثانية، أما طريقة الوصول protected فهي تعني "محمي"، أي أن الطريقة أو متغير الصنف إذا عرف protected فهذا يعني أنه لا يمكن الوصول إليه إلا من خلال الصنف الذي عرفت فيه أو الاصناف المشتقة من هذا الصنف. والمثال (٣-٢) يبين كيفية انشاء كائن من الصنف Account الذي تم تعريفه في المثال (٣-١).

مثال (٣-٢)

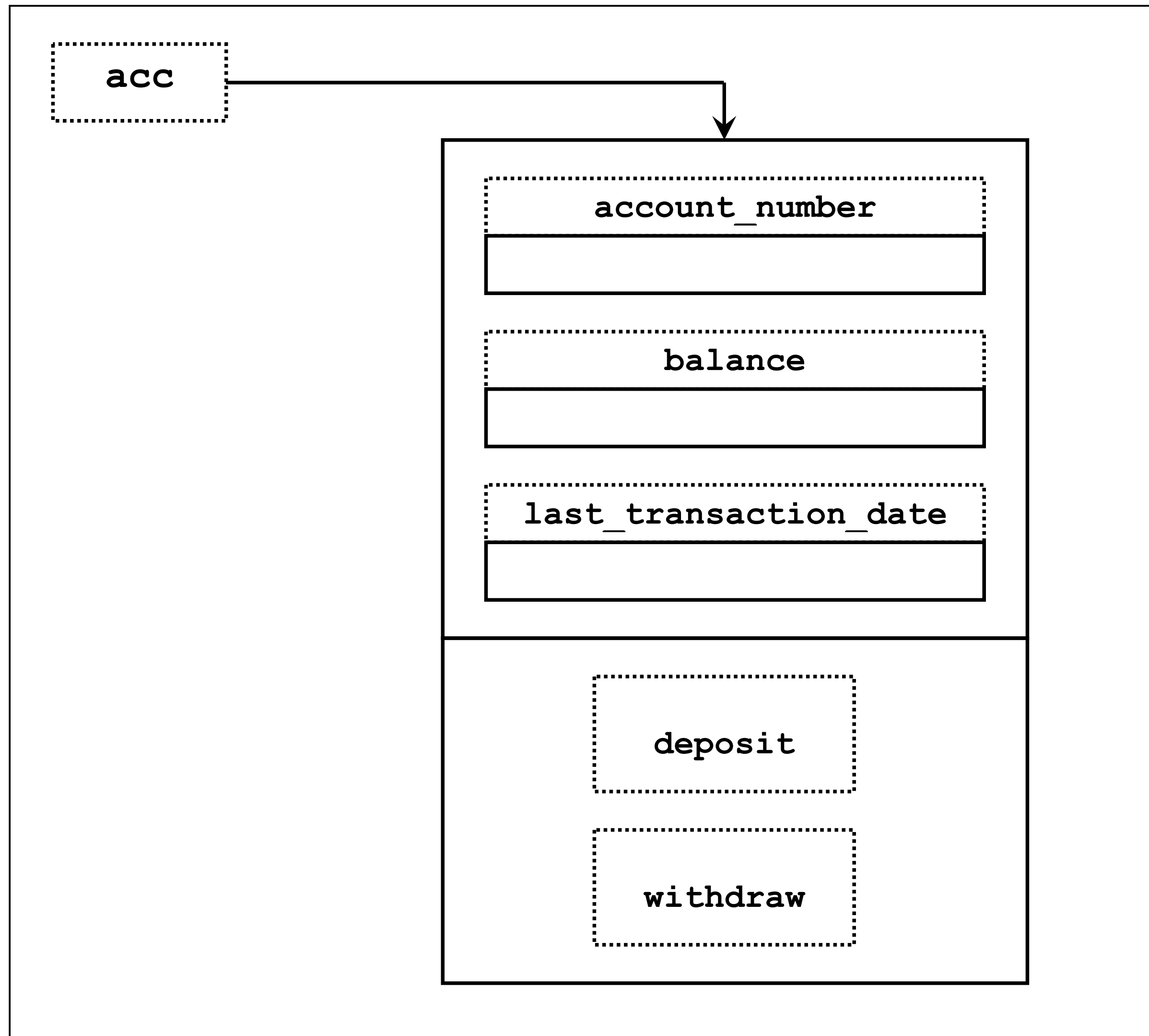
```
// Bank.java

1.  public class Bank {
2.   public static void main(String[] args) {
3.    Account acc =new Account();
4.    Acc.deposit(1000);
5.   }
6.  }
```

شرح المثال:

في السطر (٣) تم تعريف وإنشاء المتغير acc ليصبح كائن من نوع الصنف Account وهذا يعني أن المتغير acc يشير إلى كائن من نوع Account و الجملة new تقوم بإنشاء هذا الكائن بعد استدعاء احدى البانيات (Constructors) الخاصة بالصنف Account (والبانيات هي عبارة عن طرق تحمل نفس اسم الصنف بحيث يتم استدعائها عن إنشاء الكائن وتأتي بعد الكلمة المحجوزة new) وحجز الأماكن

اللازمة في الذاكرة لجميع المتغيرات. وفي السطر (٤) تمت عملية استدعاء الطريقة deposit في داخل الكائن acc، وذلك بكتابة اسم الكائن متبوعاً باسم الطريقة بحيث يفصل بينهما نقطة. والشكل (٣-٣) يبين محتويات الكائن acc.



شكل (٣-٣)

والبانيات (Constructors) كما ذكرنا سابقا هي عبارة عن طرق لها نفس اسم الصنف الذي عرّفت فيه، بحيث تستدعى عند إنشاء كائن من نوع هذا الصنف، وعندما تستدعى هذه البانيات فإنها تقوم بحجز مكان لهذا الكائن في الذاكرة وعادة ما تستخدم البانيات لإعطاء قيماً ابتدائية لمتغيرات الصنف، ويمكن للصنف الواحد أن يحتوي على أكثر من بانية وهذا ما يسمى بالتحميل الزائد للبانيات

(Overloaded Constructors)، وإذا لم نقم بتعريف بانية داخل الصنف فإنه يتم إنشاء البانية التلقائية (Default Constructor).

مثال (٣-٣):

```
// Account.java

import java.util.Date;
import javax.swing.JOptionPane;
 class Account{
  private int account_no;
  private String customer_name;
  private double balance;
  Date last_Transaction_Date;

 Account(int  no ,String name ){
  account_no=no;
  customer_name=name;
 }

 Account(int no ,String name ,double amt ){
  account_no=no;
  customer_name=name;
  balance=amt;
 }

 void deposit (double amt) {
  if (amt>0 ){
   balance += amt;
   last_Transaction_Date= new Date();
  }
  else
   JOptionPane.showMessageDialog(null,"the deposit amount  must
                                          be > 0");
 }

 void withdraw(double amt){
```

```
    if (amt<=balance ){
     balance-=amt;
     last_Transaction_Date= new Date();
    }
    else
     JOptionPane.showMessageDialog(null,"the withdraw amount
                                          must be <= balance");
   }

   public double getBalance(){
    return balance;
   }

   public String getCustomer(){
    return customer_name;
   }
  }
```

الملف الرئيسي القابل للتنفيذ حيث يحتوي على الصنف الذي بداخله الطريقة `main()`

```
// client_account.java
  public class client_account{
   public static void main(String args[]){
    Account acc1=new Account(12, "Ali");
    Account acc2=new Account(12, "Fahad", 7350.3);

    acc1.deposit(2341.5);
    acc2.withdraw(200);

    System.out.println("\n Name: "+acc1.getCustomer());
   System.out.println("\tHis Balance= " + acc1.getBalance());
   System.out.println("\tThe date of the last transaction is: " +
                                    acc1.last_Transaction_Date);

   System.out.println(" Name: "+acc2.getCustomer());
```

```
System.out.println("\tHis Balance= " + acc2.getBalance());
System.out.println("\tThe date of the last transaction is: " +
                                    acc2.last_Transaction_Date);
 System.out.println();
 }
}
```

شرح المثال:

يتكون البرنامج في هذا المثال من صنفين منفصلين، الصنف الأول Account والمحفوظ في ملف اسمه Account.java، والصنف الثاني client_account والمحفوظ في الملف المسمى client_account.java وهو الصنف الرئيسي حيث يحتوي على الطريقة main() والمعرفة داخل الصنف client_account والذي تم فية تعريف كائنين من نوع الصنف Account هما acc1 و acc2.

في الصنف الأول Account.java في الأسطر (٤-٧) تم تعريف متغيرات الصنف، ثلاثة منها عرّفت بمحدد الوصول private والذي يمنع استخدام هذه المتغيرات الثلاثة بشكل مباشر خارج هذا الصنف. في الأسطر (٨-١١) والاسطر (١٢-١٦) تم تعريف بانيان بنفس الاسم لاكن يختلفات بعدد المعاملات الشكلية. حيث يمثّل هذا الصنف "حساب بنكي" ويحتوي على ما يلي:

-البيانات (Data):

١- رقم الحساب (account_no).
٢- اسم الشخص الذي يملك هذا الحساب (customer_name).
٣- رصيد الحساب (balance).
٤- تاريخ آخر عملية تمت على الحساب (last_Transaction_Date).

-البانيات (Constructors):

١ - باني لإنشاء حساب برقم الحساب واسم صاحب الحساب:
Account(int no, String name)

٢ - باني لإنشاء حساب برقم الحساب واسم صاحب الحساب ورصيد ابتدائي:
Account(int no, String name, double amt)

-العمليات (Methods):

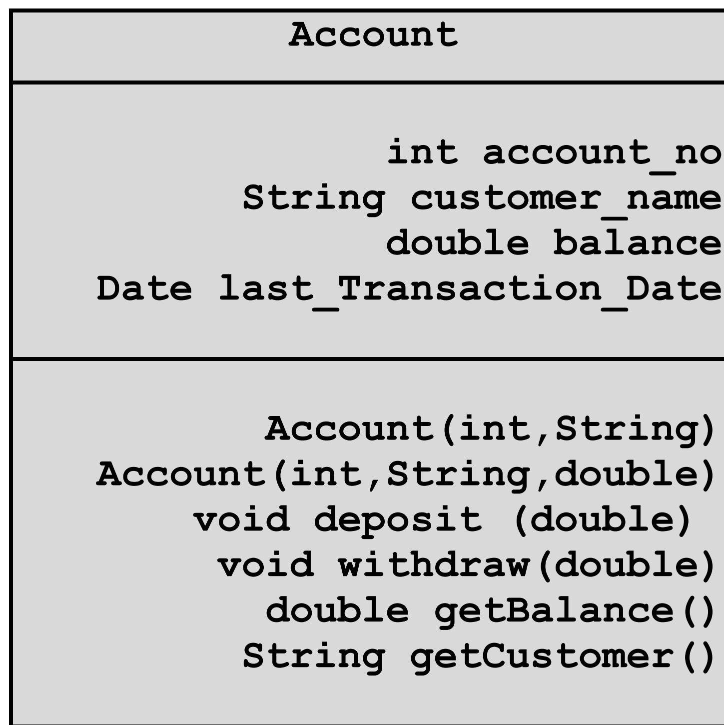
١ - عملية السحب، يجب أن يكون المبلغ المسحوب أقل أو يساوي الرصيد الحالي (deposit).

٢ - عملية الإيداع، يجب أن يكون المبلغ المودع أكبر من صفر (withdraw).

٣ -معرفة الرصيد الحالي (getBalance).

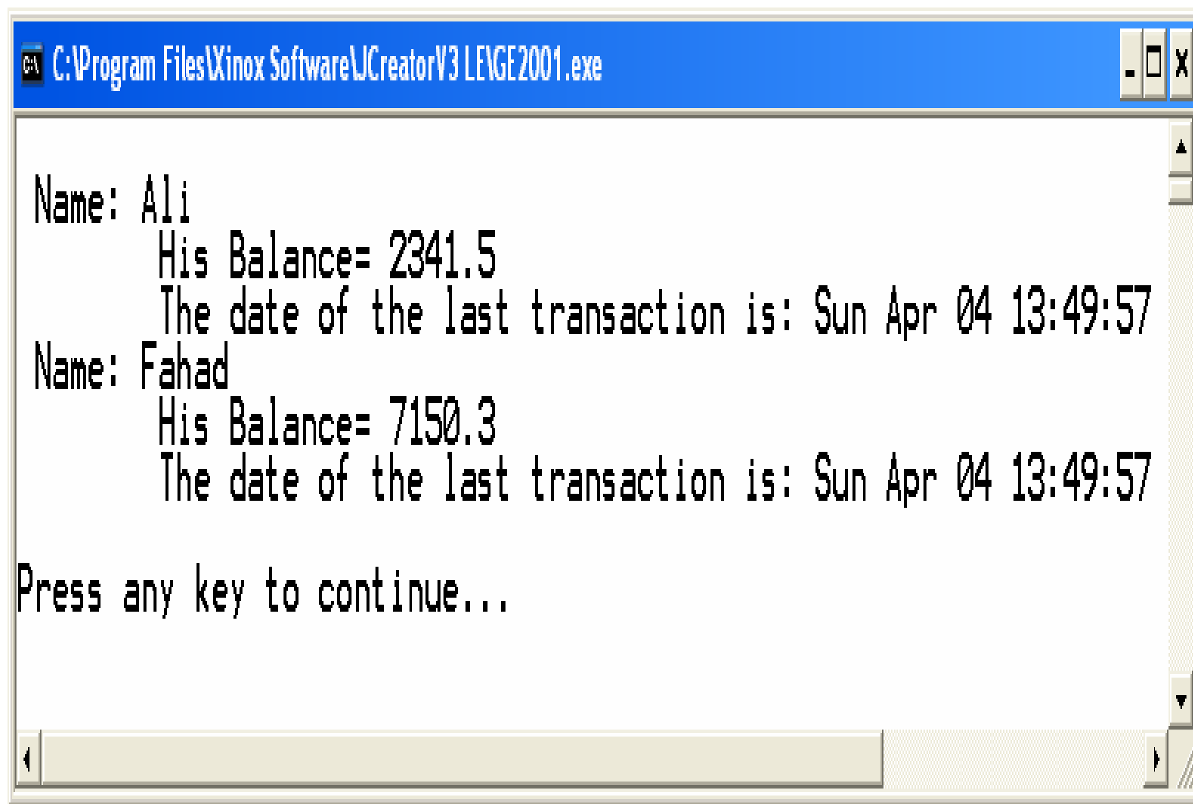
٤ -استرجاع اسم صاحب الحساب (getCustomer).

والشكل (٣-٤) يبين الشكل الرسومي للصنف Account.



شكل (٣-٤)

في الصنف الثاني (الرئيسي) client_account في الاسطري (٣-٤) تم إنشاء كائنين من نوع الصنف Account وهما acc1 و acc2، حيث تم استدعاء الباني الذي يحتاج إلى معاملين فعليين للكائن acc1 والباني الذي يحتاج إلى ثلاث معاملات فعليه للكائن acc2. وفي الاسطري (٥-٦) تم استدعاء طرق تابعة لكل من الكائنين. ومن خلال الاسطر (٩ و ١١) تم الوصول لمحتويات متغيرات الصنف المسموح الوصول إليها. والشكل (٣-٥) يبين مخرجات البرنامج في المثال السابق.



شكل(٣-٥)

وليتمكن المبرمج من إعادة استخدام الأصناف التي كتبها سابقاً دون الحاجة إلى إعادة كتابتها من جديد، لابد من وضع هذه الأصناف في حزمة (Package)، والحزمة (Package) هي عبارة عن حاوية (container) تحتوي على مجموعة من الأصناف المترابطة مع بعضها البعض ترابطا منطقيا. ومن فوائد استخدام الحزم أيضا إمكانية استخدام نفس الاسم لأكثر من صنف حيث أنه يمكن أن يكون لدينا الكثير من الأصناف التي تحمل نفس الاسم فيمكن أن نضع هذه الأصناف في حزم مختلفة وبالتالي يمكن استخدام أكثر من صنف يحمل نفس الاسم في مكان واحد.

وتتم عملية إنشاء الأصناف داخل حزمة وذلك بوضع الكلمة المحجوزة package في بداية الملف الذي يحتوي تعريف الصنف او الأصناف متبوعة باسم الحزمة، وبعد عملية الترجمة الناجحة للبرنامج يتم تخزين الأصناف وتحديداً الملفات ذات الإمتداد class في هذه الحزمة. وإذا لم تكن هذه الحزمة موجودة يتم إنشاؤها بعد انتهاء عملية الترجمة. والمثال (٣-٤) يوضّح عملية إنشاء الحزم (Packages).

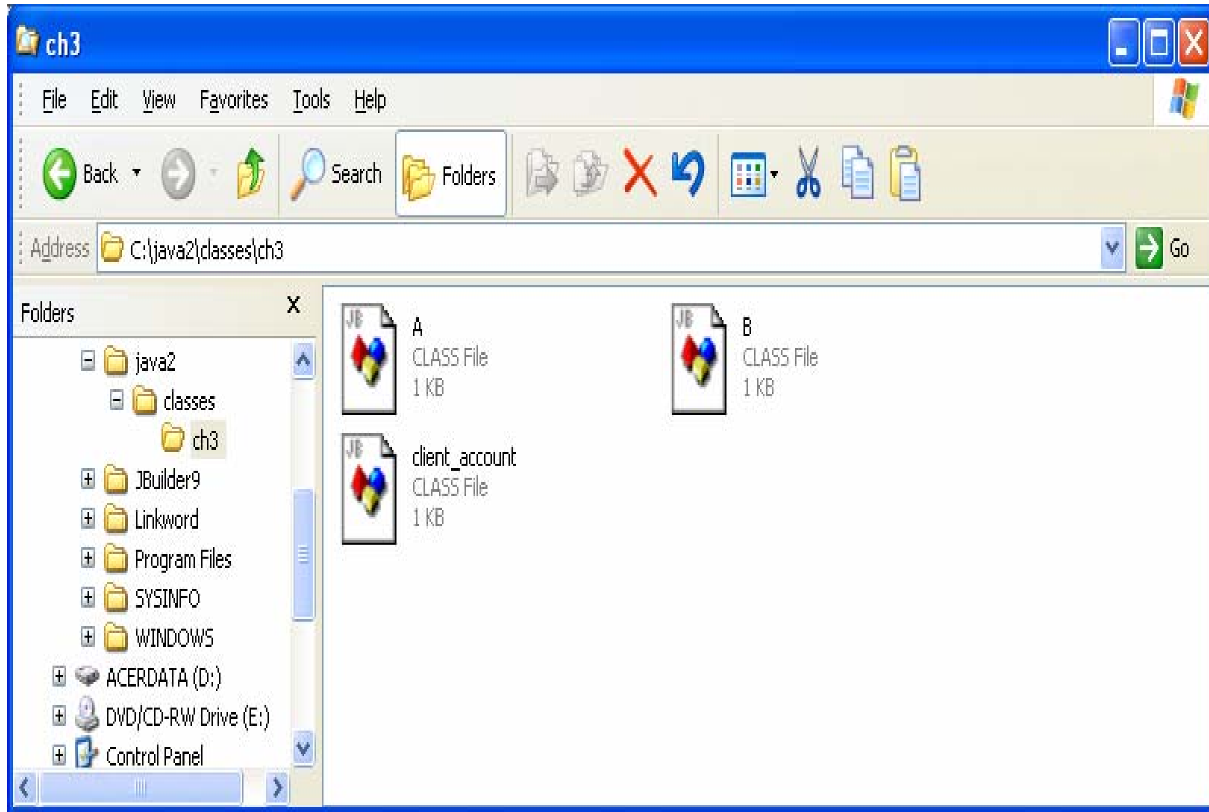
مثال (٣-٤):

```
// client_account.java

1.     package java2.classes.ch3;
2.
3.     class A{
4.       .
5.       .
6.       .
7.     }
8.
9.     class B{
10.      .
11.            .
12.      .
13.    }
14.
15.     public class client_account{
16.             public static void main(String args[]){
17.              .
18.       .
19.       .
20.             }
21.    }
```

شرح المثال:

في السطر (١) تم تحديد اسم ومسار الحزمة (Packages) التي ستحوي على جميع الأصناف التي سيتم إنشائها بعد ترجمة هذا المثال. بحيث سيتم تخزين الأصناف: A.class، B.class، و client_account.class في المجلد ch3 الموجود داخل المجلد classes والموجود داخل المجلد java2. لاحظ في هذا المثال وجود أكثر من صنف في نفس الملف وبعد الترجمة سيتحول كل صنف إلى ملف منفصل ذو امتداد class، لكن يجب أن لا يحتوي الملف الواحد على أكثر من صنف معرف كصنف عام public. والشكل (٣-٦) يبين الأصناف والمجلدات بعد ترجمة هذا المثال.



شكل (٣-٦)

مفهوم الوراثة (Inheritance):

تتم عملية الوراثة بين الأصناف من خلال اشتقاق صنف من صنف آخر، ففي هذه الحالة يرث الصنف المشتق (SubClass) جميع محتويات الصنف المشتق منه (Super Class) باستثناء المحتويات المعرّفة على أنها خاصة (private). وتتم عملية اشتقاق الأصناف من بعضها باستخدام الكلمة المحجوزة extends. والمثال (٣-٥) يبين عملية اشتقاق صنف من صنف آخر. ومن الجدير بالذكر أن لغة جافا لا تدعم الوراثة المتعددة، أي أن الصنف لا يمكن أن يرث أكثر من صنف واحد فقط.

مثال (٣-٥):

```
// y.java
class x{
 .
 .
 .
} // end of class x
public class y extends x{
 .
 .
 .
}
```

شرح المثال:

في هذا المثال تم تعريف الصنف x ومن ثم تم تعريف الصنف y المشتق من الصنف x والذي سيرث كل محتوياته، ونستطيع أن نقول أن الصنف x هو الأب والصنف y هو الابن، حيث سيرث الابن بعض أو كل صفات الأب. وفي هذا المثال التوضيحي لم نعرّف محتويات أي من الصنفين.

الوصول للطرق والبيانات الموروثة:

يتم الوصول للطرق والبيانات (المخزنة في متغيرات الصنف) بشكل مباشر داخل الصنف المشتق وكأنها معرفّة داخلة. ويجب أن نتذكر باننا لا نستطيع الوصول للطرق والمتغيرات المعرفة على أنها خاصة (private) بالصنف الذي عرّفت فيه، حيث أنها لا تورث للأصناف المشتقة من هذا الصنف. والمثال (٢-٦) يبين كيفية الوصول للطرق والبيانات (المتغيرات) الموروثة.

مثال (٣-٦):

```
// Cars.java

1.    class Transportation{
2.     protected static int x=12;
3.     private int y=19;
4.     public static void meth1(){
5.      System.out.println("Calling meth1() from class Cars.");
6.     }

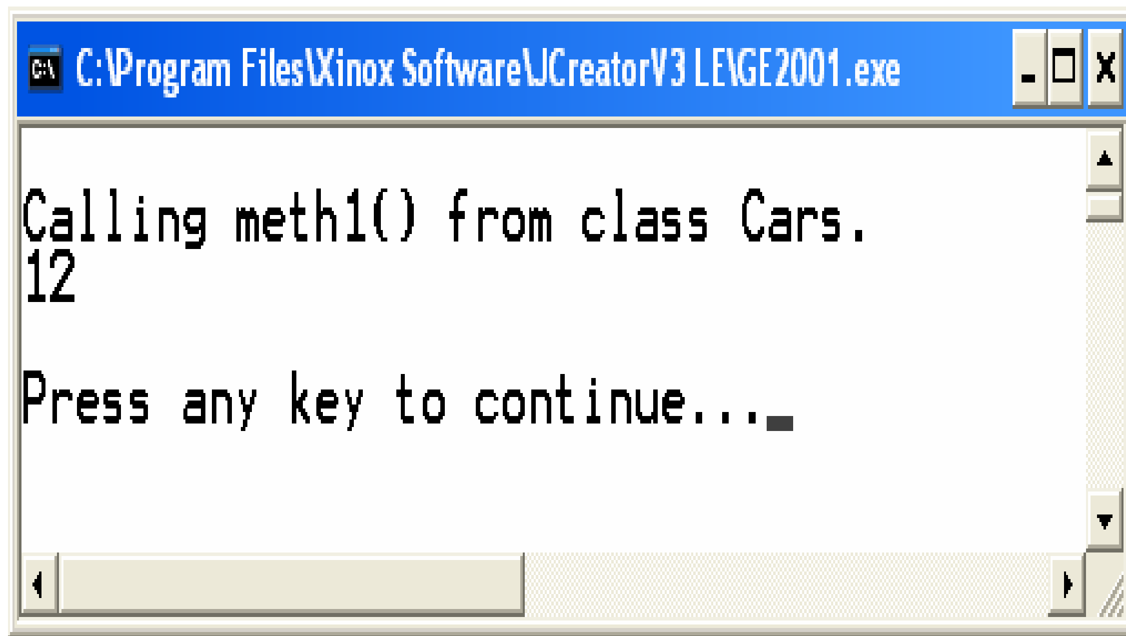
7.     private void meth2(){
8.      System.out.println("will not be called from Cars");
9.     }
10.   } // end of class Transportation

11.   public class Cars extends Transportation{
12.    public static void main(String args[]){
13.     meth1();
14.     System.out.println(x);
15.     //     meth2(); // meth2() has private access in
              // Transportation
16.     // System.out.println(y); // y has private access
                       // on Transportation
17.    }
18.   } // end of class Cars
```

شرح المثال:

في الأسطر (١-١٠) تم تعريف الصنف Transportation ليحتوي على متغيرين صنف هما: المتغير x وهو متغير محمي (protected) والمتغير y وهو متغير خاص (private)، ويحتوي هذا الصنف أيضاً على طريقتين هما: الطريقة meth1() وهي طريقة عامة (public) والطريقة meth2() وهي طريقة خاصة (private). وفي الاسطر (١٢-١٩) تم تعريف صنف ثاني هو الصنف Cars، بحيث تم اشتقاق هذا الصنف Cars من الصنف Transportation، وفي هذه الحالة يعتبر الصنف Transportation هو الـــ

Super Class و الصنف Cars هو الـــ SubClass، ومن منظور آخر نستطيع القول بأن الصنف Transportation هو الأب والصنف Cars هو الأبن. وفي هذا المثال يتم توريث جميع صفات (وهذه الصفات هي متغيرات الصنف و الطرق) الاب Transportation إلى الابن Cars باستثناء الصفات المعرفة على أنها خاصة private. وفي الاسطري (١٤-١٥) تم استخدام المتغير x والطريقة meth1() والمعرفين في صنف الأب (Transportation) وذلك لأنه تم توريثهم لصنف الابن (Cars)، بينما في الاسطري (١٦-١٧) لم نستطع استخدام المتغير y والطريقة meth2() لانهما معرفان بكونهما خاصين (private) بالصنف الذي تم تعريفهم فيه (Transportation)، وعند محاولة استخدامهم يظهر خطأ كما هو موضّح في المثال السابق. والشكل (٣-٧) يبين مخرجات هذا المثال.



شكل (٣-٧)

وعندما تقوم الطريقة method المعرفة داخل الصنف باستخدام احد مكونات هذا الصنف ففي بعض الأحيان لا يوجد هناك ما يثبت أن هذه العملية تجري على المكون الصحيح، لذلك فإن لغة جافا توفر المؤشر this الذي يشير إلى النسخة الحالية من الصنف وبالتالي فإن استخدام this يضمن أن تتم العملية على المكون الصحيح . والمثال (٣-٧) يوضّح كيف يتصرف البرنامج بدون استخدام المؤشر this، بينما يوضّح المثال (٣-٨) الفائدة من استخدام المؤشر this.

مثال (٣-٧):

```
// C.java

1.   class A{
2.     protected int a=9;
3.   } // end of class A

4.    class B extends A{
5.     void test(){
6.             int a=22;
7.             System.out.println("a = "+a);
8.     }
9.   } // end of class B

10.  public class C{
11.    public static void main(String args[]){
12.            B acc=new B();
13.            acc.test();
14.    }
15.  } // end of class C
```

شرح المثال:

تكون مخرجات هذا المثال هي طباعة ما يلي: a=22، وذلك لأن الجملة في السطر (٧) تتعامل مع المتغير a التابع للطريقة test() وليس المتغير a التابع للصنف A.

مثال (٣-٨):

```
// C.java

class A{
  protected int a=9;
} // end of class A

class B extends A{
  void test(){
```

```
6.            int a=22;
7.            System.out.println("a = "+this.a);
8.     }
9.   } // end of class B

10.  public class C{
11.    public static void main(String args[]){
12.            B acc=new B();
13.            acc.test();
14.    }
15.  } // end of class C
```

شرح المثال:

بينما تكون مخرجات هذا المثال هي طباعة ما يلي: a=9، وذلك لأن الجملة في السطر (٧) تتعامل مع المتغير a التابع للصنف A وليس المتغير a التابع للطريقة test()، وذلك باستخدام المؤشر this والذي حدد أن المتغير المقصود هو المتغير a الموجود في الصنف الحالي B وبما أن الصنف B لا يحتوي على متغير باسم a فلذلك يتم استخدام المتغير a التابع للصنف A الذي تم اشتقاق الصنف الحالي منه.

بينما إذا احتوى الصنف المشتق (SubClass) والصنف المشتق منه (Super Class) على متغير أو طريقة بنفس الاسم، ففي هذه الحالة سواء تم استخدام المؤشر this أو لم يتم أستخدامة داخل الصنف المشتق (SubClass) فإنه يتم الرجوع للطريقة أو المتغير التابع لهذا الصنف المشتق (SubClass) ولن يتم الرجوع بأي حال من الأحوال إلى الطريقة أو المتغير المعرّف داخل الصنف المشتق منه (Super Class). وعند الحاجة للرجوع من داخل الصنف المشتق (SubClass) للطريقة أو المتغير التابع للصنف المشتق منه (Super Class) مع وجود تعريف لطريقة أو لمتغير بنفس الاسم داخل الصنف الحالي (SubClass) لابد من استخدام المؤشر super والذي يمكّننا من استخدام محتويات الصنف المشتق منه (Super Class). والمثال (٣-٩) يوضّح الفرق بين استخدام وعدم استخدام المؤشر super، حيث إن نتائج التنفيذ تختلف في كل حالة من الحالات.

مثال (٣-٩):

```
// C.java

1.	class A{
2.		void test1(){
3.	 System.out.println("The test1() method was invoked FROM class
					A");
4.		}
5.	} // end of class A

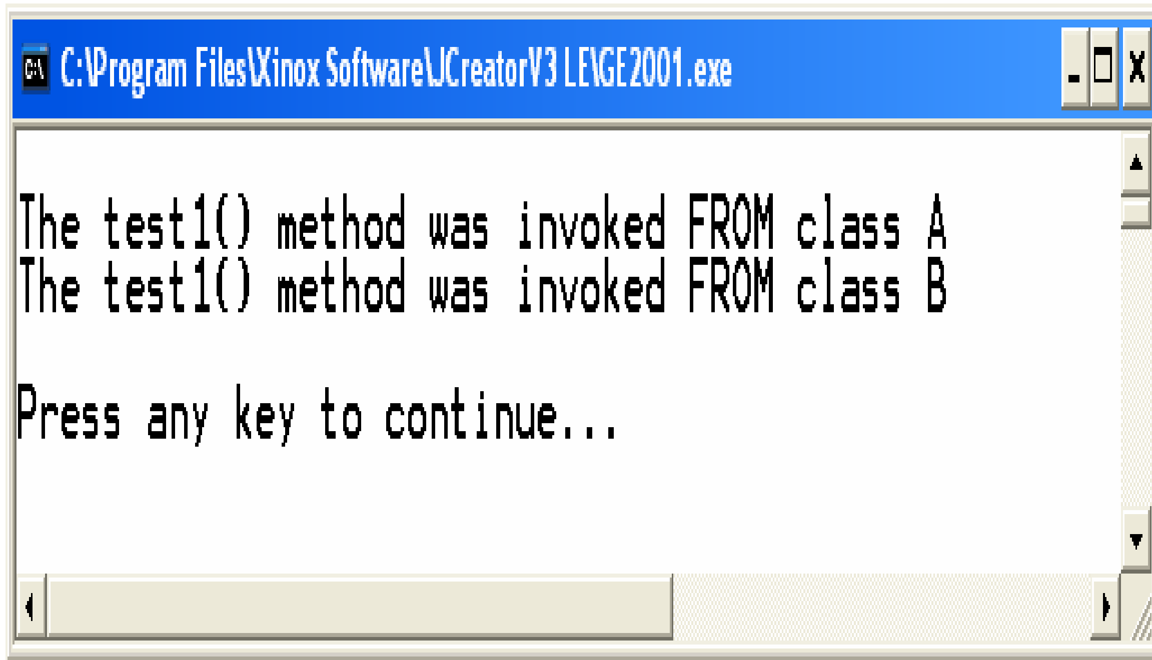
6.	class B extends A{
7.		void test(){
8.		 super.test1();
9.		 test1();
10.		}
11.		void test1(){
12.	 System.out.println("The test1() method was invoked FROM class
					B");
13.		}
14.	} // end of class B

15.	public class C{
16.		public static void main(String args[]){
17.		 B acc=new B();
18.		 acc.test();
19.		}
20.	} // end of class C
```

شرح المثال:

في السطر (٩) الموجود في الصنف B المشتق من الصنف A، تم استدعاء الطريقة test1() والمعرّفة في كل من الصنف B والصنف A. وفي هذا السطر تم تحديد استدعاء الطريقة test1() والمعرّفة في الصنف المشتق منه A، وتم هذا التحديد عن طريق استخدام المؤشر Super. بينما في السطر (١٠) تم استدعاء الطريقة test1() دون استخدام المؤشر Super وبالتالي سيتم تنفيذ الطريقة test1() المعرفة داخل الصنف

الحالي (الصنف الذي تمت عملية الاستدعاء منه) B. والشكل (٣-٨) يبين المخرجات عند تنفيذ البرنامج المكتوب في هذا المثال.



شكل(٣-٨)

استبدال الطرق الموروثة (Methods Overriding):

قد نحتاج في بعض البرامج إلى استبدال وتغيير طبيعة عمل الطرق الموروثة من صنف الاب (Super Class) إلى صنف الابن (SubClass)، ففي هذه الحالة لابد لنا من استبدال الطريقة الموروثة (Methods Overriding) وذلك بإعادة تعريف هذه الطريقة بالشكل الذي نريد، وإذا احتجنا إلى الوصول للطريقة الموروثة لابد من استخدام المؤشر super كما لاحظنا سابقاً من هذه الوحدة.

مثال (٣-١٠):

```
// Test.java

1.    class Car {
2.     private int year;
3.     private float originalPrice;
```

```
  // calculate the sale price of a car based on its
  // cost
  public double CalculateSalePrice() {
   double salePrice;
   if (year > 1994)
    salePrice = originalPrice * 0.75;
   else if (year > 1990)
    salePrice = originalPrice * 0.50;
   else
    salePrice = originalPrice * 0.25;
   return salePrice;
  }

  // a public constructor
  public Car(int year, float originalPrice) {
   this.year = year;
   this.originalPrice = originalPrice;
  }
 }

 class ClassicCar extends Car {
  // calculate the sale price of a car based on its
  // cost
 public double CalculateSalePrice() {
   return 10000;
  }

  // a public constructor
  public ClassicCar(int year, float originalPrice) {
   super(year, originalPrice);
  }
 }

 public class Test{
  public static void main(String args[]){
   ClassicCar myClassic = new ClassicCar(1920, 1400);
   double classicPrice =
```

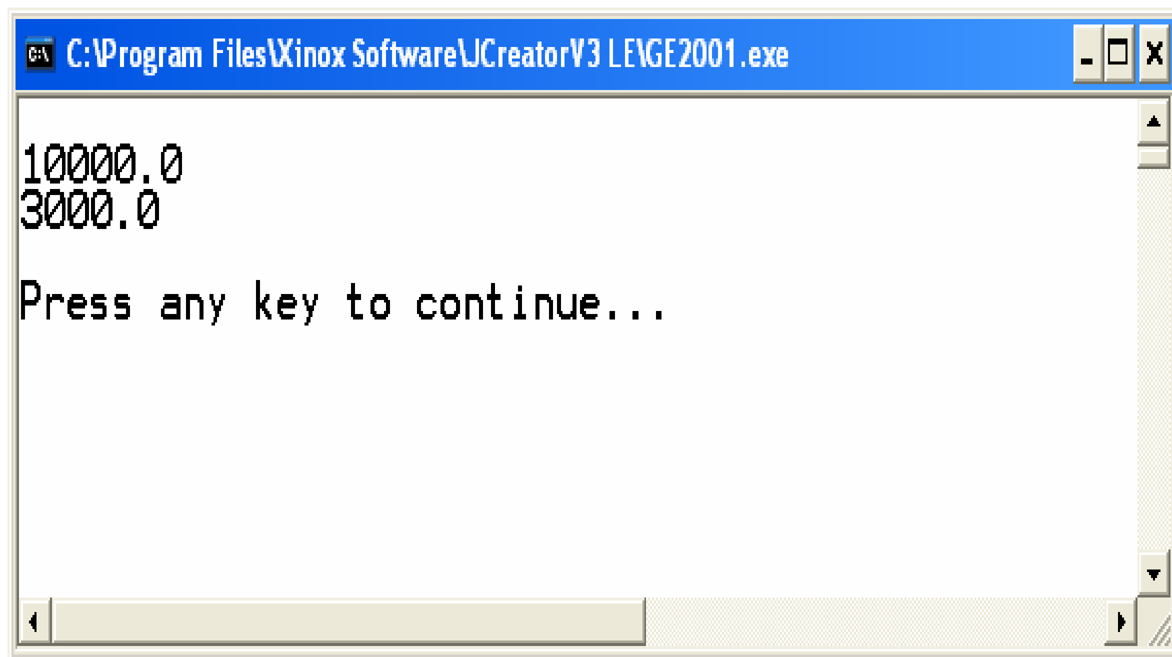
```
                myClassic.CalculateSalePrice();
35.     System.out.println(classicPrice);

36.     Car myCar = new Car(1990, 12000);
37.     double price = myCar.CalculateSalePrice();
38.     System.out.println(price);
39.   }
40.  }
```

شرح المثال:

في ألاسطر (٥-١٤) تم تعريف الطريقة ()CalculateSalePrice بالنسبة للصنف Car، وفي الصنف ClassicCar المشتق من الصنف Car احتجنا إلى تغيير عمل الدالة الموروثة ()CalculateSalePrice مما دعانا إلى إعادة تعريفها بالنسبة للصنف ClassicCar، وتمت عملية إعادة التعريف في الأسطر (٢٣-٢٥) وهذا مايسمى باستبدال الطرق الموروثة (Methods Overriding). لاحظ في السطر (١٧) اضطررنا إلى استخدام المؤشر this وذلك للتميز بين المعامل الشكلي year والمتغير year الموروث من الصنف Car.

والشكل (٣-٩) يبين مخرجات البرنامج في هذا المثال.



شكل (٣-٩)

تمارين:

س١: أنشئ صنف وسمه Rational لتنفيذ العمليات الحسابية على الكسور، بحيث يحتوي هذا الصنف على ما يلي:

-متغيرات الصنف: متغيرين خاصين (private) من نوع int هما: numerator (ليحتوي على البسط) و denominator (ليحتوي على المقام).

-باني (Constructor) يسمح بإعطاء قيمة ابتدائية (يجب أن يتم تخزين القمية الابتدائية للكسر بشكل مختصر، مثلاً: يتم تخزين 1/2 بدلاً من 2/4) للكسر عند تعريف كائن (Object) من نوع هذا الصنف.

-طريقة لجمع رقمين كسريين وتخزين الناتج بشكل مختصر.

- طريقة لطرح رقمين كسريين وتخزين الناتج بشكل مختصر.

-طريقة لضرب رقمين كسريين وتخزين الناتج بشكل مختصر.

-طريقة لقسمة رقمين كسريين وتخزين الناتج بشكل مختصر.

- طريقة لطباعة الرقم الكسري بالشكل التالي: a/b حيث a تمثل البسط (numerator) و b تمثل المقام (denominator).

- طريقة لطباعة الرقم الكسري على شكل رقم حقيقي (float)، وذلك بقسمة البسط على المقام.

ثم اكتب برنامج يستخدم هذا الصنف.

س٢: أنشئ صنف يمثل مربع وسمه Rectangle، بحيث يحتوي على المتغيرات التالية: length و width يأخذ كل واحد منهم قيمة ابتدائية تساوي ١، ويحتوي على طريقتين، تقوم الطريقة الأولى بحساب مساحة المربع، بينما تقوم الطريقة الثانية بحساب محيط المربع. ثم اكتب برنامج لتطبيق هذا الصنف.

س٣: أنشئ صنف مسمى HugeInteger لتمثيل عدد صحيح كبير جداً وذلك باستخدام مصفوفة مكونه من أربعين موقع كل موقع يحتوي على خانة واحدة من خانات هذا الرقم. ويحتوي هذا الصنف على الطرق التالية، (اكتب برنامج لتطبيق هذا الصنف):

-الطريقة inputHugeInteger لإدخال الرقم الصحيح الكبير جداً إلى المصفوفة.

- الطريقة outputHugeIntege لطباعة الرقم الصحيح الكبير جداً على الشاشة.
- الطريقة addHugeIntege لجمع عددين صحيحين كبيرين جداً.
- الطريقة subtractHugeIntege لطرح عددين صحيحين كبيرين جداً.
-الطريقة isEqualTo للسؤال فيما إذا كانا عددين صحيحين كبيرين جداً متساويين أم لا ، بحيث ترجع true إذا كانا متساويين وترجع false إذا كانا غير متساويين.
- الطريقة isGreaterThan للمقرنة بين العددين الصحيحين وتحيد هل الاول أكبر من الثاني أم لا، بحيث ترجع true إذا كان العدد الصحيح الاول أكبر من العدد الصحيح الثاني وترجع false إذا كان غير ذلك.

س٤: أنشئ صنف لتمثيل التاريخ بحيث يسمى Date وله الخصائص التالية:

-يتم إخراج التاريخ بأحد الأشكال التالية:

MM/DD/YYYY
April 01, 2004
DDD YYYY

-استخدم التحميل الزائد للبانيات لإنشاء كائنات تحمل قيماً ابتدائية للتاريخ حسب أشكال التاريخ السابقة.

ثم اكتب برنامج يستخدم كائنات من نوع هذا الصنف.

س٥: أنشئ صنف يسمى IntegerSet، بحيث يحتوي كل كائن من هذا الصنف على أرقام صحيحة بين الصفر و ١٠٠، ويتم تمثيل مجموعة الأعداد الصحيحة داخلياً كمصفوفة من نوع boolean، وعندما تكون قيمة عنصر المصفوفة a[i] مساوية لِ true يكون الرقم i من عناصر المجموعة، بينما إذا كانت قيمة العنصر a[j] تساوي false يكون الرقم j غير موجود في هذه المجموعة. ويقوم الباني الذي ليس له عوامل شكلية بإنشاء مجموعة فارغة، أي أن جميع عناصر المصفوفة تحتوي على القيمة false. كما ويحتوي هذا الصنف على الطرق التالية:

-الطريقة unionOfIntegerSet تنشئ مجموعة ثالثة تحتوي على ناتج تنفيذ عملية الاتحاد بين مجموعتين.

- الطريقة intersectionOfIntegerSet تنشئ مجموعة ثالثة تحتوي على ناتج تنفيذ عملية التقاطع بين مجموعتين.
- الطريقة insertElement تضيف العنصر k إلى المجموعة.
- الطريقة deleteElement تحذف العنصر m من المجموعة.
-الطريقة setPrint تطبع محتويات المجموعة وتطبع "Empty Set" إذا كانت المجموعة فارغة.
-الطريقة isEqualTo تقارن بين مجموعتين فيما إذا كانا متساويين أم لا.

اكتب برنامج يستخدم كائنات من نوع هذا الصنف.

المراجع

١. Deitel and Deitel, Java: How to Program, 3rd Edition, Prentice Hall, 2001

٢. Patrick Naughton and Michael Morrison, the Java Handbook, McGraw-Hill, 1996

٣. Bruce Eckel, Thinking in Java (2nd Edition), 2001

٤. م. فادي حجار، لغة البرمجة JAVA 2، دار شعاع للنشر والعلوم، ٢٠٠١

٥. كن أرنولد، لغة برمجة جافا، مركز التعريب والبرمجة، ٢٠٠١

المحتويات

الصفحة

تقدر المؤسسة العامة للتعليم الفني والتدريب المهني الدعم

المالي المقدم من شركة بي آيه إي سيستمز (العمليات) المحدودة

GOTEVOT appreciates the financial support provided by BAE SYSTEMS

BAE SYSTEMS